LIMBAJUL PASCAL

Corina Corici
Dorin Mânz
Adriana Simulescu
Marinel Şerban

LIBRIS

LIMBAJUL PASCAL

Corina Corici
Dorin Mânz
Adriana Simulescu
Marinel Şerban

LIBRIS

CLUJ 1991

COORDONATOR: CLARA IONESCU

Control ştiinţific: lect.dr.**FLORIAN BOIAN**
prof.**DOINA RANCEA**

Cartea a fost elaborată astfel:

Corina Corici: cap.II,III
Dorin Mânz: cap.VI
Adriana Simulescu: cap.IV
Marinel Şerban: cap.I,V,VII,VIII

Redactor: **Clara Ionescu**
Tehnoredactare computerizată: **Cristina Tămaş**
Grafica: **Baka József**
Grafica computerizată: **Eugen Ionescu şi Adrian Hosu**
Coperta: **Sipos László**

Bun de tipar: 8.05.1991.
Coli de tipar 14
ISBN 973-95118-0-5

CUPRINS

CAPITOLUL I. NOTIUNI INTRODUCTIVE

1.1. Evoluţia limbajelor de programare. Principii ale programării structurate..............5
1.2. Structura programelor PASCAL.......................7
1.3. Descrierea sintaxei limbajului PASCAL cu ajutorul diagramelor..........................8

CAPITOLUL II. ELEMENTE DE BAZA ALE LIMBAJULUI PASCAL

2.1. Vocabularul limbajului PASCAL.....................11
2.1.1. Setul de caractere........................11
2.1.2. Identificatori............................11
2.1.3. Separatori şi comentarii..................12
2.2. Constantele limbajului PASCAL.....................13
2.2.1. Constante întregi.........................13
2.2.2. Constante reale...........................13
2.2.3. Şir de caractere..........................14
2.2.4. Constante desemnate prin identificatori....14
2.3. Definirea constantelor utilizator.................15
2.4. Noţiunea de tip...................................15
2.5. Variabilele limbajului PASCAL şi declararea lor...16
2.6. Expresii. Funcţii standard........................16
2.7. Operaţii de citire/scriere simple.................19

CAPITOLUL III. STRUCTURI FUNDAMENTALE DE CONTROL IN LIMBAJUL PASCAL

3.1. Structura secvenţială.............................23
3.1.1. Instrucţiunea de atribuire................23
3.1.2. Instrucţiunea compusă.....................24
3.2. Structura alternativă.............................26
3.2.1. Instrucţiunea IF..........................26
3.2.2. Instrucţiunea CASE........................30
3.3. Structura repetitivă..............................34
3.3.1. Instrucţiunea WHILE.......................35
3.3.2. Instrucţiunea REPEAT......................37
3.3.3. Instrucţiunea FOR.........................40
3.4. Instrucţiunea GOTO................................43

CAPITOLUL IV. TIPURI DE DATE

4.1. Tipuri simple.....................................48
4.1.1. Tipuri simple standard....................49
4.1.2. Tipuri simple definite de utilizator.......57
4.2. Tipuri structurate................................60
4.2.1. Tipul tablou..............................60
4.2.2. Tipul înregistrare........................67
4.2.3. Tipul mulţime.............................71

CAPITOLUL V. SUBPROGRAME

5.1. Domeniul de valabilitate al identificatorilor şi

etichetelor....................................76
5.2. Dezvoltarea programelor PASCAL....................77
5.3. Proceduri..79
5.3.1. Declararea procedurilor. Parametri formali, parametri efectivi......79
5.3.2. Proceduri standard de intrare/ieşire.......81
5.3.3. Reprezentarea datelor în memorie...........93
5.3.4. Accesul direct al memoriei.................94
5.3.5. Proceduri HP4TM pentru lucrul cu caseta....98
5.3.6. Alocarea dinamică a memoriei..............100
5.3.7. Proceduri definite în program.............100
5.4. Funcţii...107
5.4.1. Declararea funcţiilor.....................107
5.4.2. Funcţii standard specifice limbajului PASCAL HP4TM............................108
5.4.3. Funcţii definite în program...............120
5.4.4. Stocarea datelor în memorie...............122
5.5. Recursivitate.....................................126
5.5.1. Generalităţi..............................126
5.5.2. Program recursiv de ilustrare a mecanismului recursivităţii...........................127
5.5.3. Utilizarea recursivităţii.................129
5.5.4. Tipuri de algoritmi recursivi.............132

CAPITOLUL VI. ALOCAREA DINAMICA A MEMORIEI

6.1. Tipul referinţă (pointer)..........................149
6.2. Structuri de date de tip listă liniară.............152
6.2.1. Implementarea listelor cu ajutorul tipului pointer (liste înlănţuite)................152
6.2.2. Tehnici de inserare a nodurilor...........153
6.2.3. Tehnici de suprimare (ştergere) a nodurilor unei liste.............................159
6.2.4. Traversarea unei liste înlănţuite..........161
6.2.5. Aplicaţii ale listelor înlănţuite..........165
6.2.6. Liste dublu înlănţuite.....................168
6.2.7. Structuri de date de tip stivă.............172
6.2.8. Structuri de date de tip coadă.............174
6.2.9. Structuri de date de tip arbore binar......176

CAPITOLUL VII. ELEMENTE DE GRAFICA ŞI SUNET SPECIFICE LIMBAJULUI PASCAL HP4TM

7.1. Generalităţi......................................194
7.2. Variabile globale.................................194
7.3. Procedurile pachetului TURTLE.....................195

CAPITOLUL VIII. MIC MANUAL DE OPERARE PASCAL HP4TM

8.1. Introducere.......................................208
8.1.1. Punerea în funcţiune......................208
8.1.2. Compilarea şi lansarea în execuţie........209
8.2. Editorul..210
8.2.1. Introducere în editor.....................210
8.2.2. Comenzile editorului......................210
8.3. Opţiuni ale compilatorului........................215

ANEXA 1. CUVINTE SI IDENTIFICATORI PREDEFINITI.............218
ANEXA 2. HP4TM -COMPILATORUL PASCAL ZX Spectrum - MEMORATOR219

Bibliografie...222

I. NOŢIUNI INTRODUCTIVE

1.1. EVOLUŢIA LIMBAJELOR DE PROGRAMARE. PRINCIPII ALE PROGRAMARII STRUCTURATE

Limbajele de programare sînt mijloace de comunicare între utilizator şi sistemul de calcul. Prin intermediul lor, omul transmite calculatorului ordinele necesare pentru executarea unei anumite operaţii. Descrierea cu ajutorul unui limbaj de programare a etapelor necesare rezolvării unei anumite probleme se numeşte *program*. Astfel, un program apare ca o succesiune de comenzi transmise calculatorului pe care acesta le execută. O asemenea comandă se numeşte *instrucţiune*. Este evident că programul transmis calculatorului pentru a fi executat nu este altceva decît descrierea, în limbajul de programare ales, a algoritmului de rezolvare a problemei respective.

Se ştie că un calculator nu "cunoaşte" decît un singur limbaj - limbajul calculatorului, numit *cod-maşină*. Primele generaţii de calculatoare erau programate direct în cod-maşină, lucru deosebit de dificil de realizat. Odată cu apariţia necesităţilor tot mai mari de prelucrare a informaţiilor şi odată cu progresele înregistrate în domeniul construcţiei de calculatoare, s-a impus un salt calitativ în ceea ce priveşte programarea calculatoarelor şi anume trecerea de la programarea în cod-maşină la *programarea simbolică*. In acest fel programatorul se preocupă mai mult de metoda de rezolvare a problemei concrete, fără a se mai preocupa de amănunte legate de calculator. In acest fel, pe lîngă o comoditate sporită în scrierea programelor, această schimbare a însemnat şi cîştigarea unei independenţe a programării faţă de caracteristicile fizice ale calculatorului.

In evoluţia limbajelor de programare simbolice există cîteva repere care trebuie menţionate: anul 1955 marchează apariţia limbajului FORTRAN, destinat calculelor tehnico-ştiinţifice cu caracter numeric (**FOR**mula **TRAN**slation = traducerea formulelor). In prezent se utilizează pe calculatoarele mari existente în ţară varianta FORTRAN IV, iar pe minicalculatoare, FORTRAN 77, care diferă mult de versiunea iniţială.

Un alt punct de referinţă în evoluţia limbajelor de programare simbolice este anul 1960; în acest an un grup de cercetători, coordonat de Peter Naur, a definit un limbaj de programare, numit ALGOL-60 (**ALGO**rithmic **L**anguage = limbaj algoritmic), a cărui definiţie se remarcă prin precizie şi o sintaxă complet formalizată. Deşi limbajul ALGOL a avut o utilizare relativ restrînsă, multe din conceptele introduse cu prilejul definirii limbajului sînt folosite şi astăzi, ALGOL-60 fiind un model pentru proiectanţii altor limbaje de programare. Tot în 1960 a apărut şi prima versiune a limbajului COBOL (**CO**mmon **B**ussiness **O**riented **L**anguage = limbaj orientat pe probleme economice).

Intre anii 1960-1970 au apărut nenumărate limbaje de programare, unele rezistînd timpului, fiind folosite mai mult sau mai puţin, altele, din contră, dispărînd la fel de rapid cum au apărut.

Dintre limbajele care au rezistat timpului, care au adus ceva nou şi care sînt folosite în prezent din ce în ce mai mult, remarcăm limbajul **PASCAL**, definit în anul **1971** de **Niklaus Wirth**.

Apariţia acestui limbaj este un rezultat al conceptelor dezvoltate ca urmare a crizei ce caracteriza domeniul programării calculatoarelor la sfîrşitul anilor '60. In această perioadă problemele abordate de către programatori au devenit tot mai complexe. Programele construite pentru rezolvarea acestor probleme au devenit şi ele, în mod evident, complicate, greu accesibile chiar şi pentru creatorii lor. Depanarea şi modificarea unor astfel de programe a devenit extrem de complicată. Acest fenomen a apărut datorită absenţei unor principii clare, care să impună o "disciplină a programării". Bineînţeles, a apărut întrebarea: *nu se poate oare elabora o tehnologie generală care să permită realizarea sistematică a unor programe elegante, uşor de depanat şi de modificat?* Ca răspuns la această întrebare a apărut **metoda programării structurate**.

Un program structurat este format din unităţi funcţionale bine conturate, ierarhizate conform naturii problemei. In interiorul unei astfel de unităţi, structurarea se manifestă atît la nivelul instrucţiunilor cît şi la nivelul datelor.

Programarea structurată este o *metodă independentă de limbajul de programare*, ea acţionînd la nivelul stilului de lucru. Metoda programării structurate implică în primul rînd principiul de programare "de sus în jos" - adică descompunerea programului în elemente logice independente, numite *module*, fiecare modul avînd o funcţie bine definită în cadrul programului.

Avantajele unei astfel de metode de lucru sînt:

a)-scrierea programului poate fi făcută de mai multe persoane, fiecare ocupîndu-se de cîte un modul;

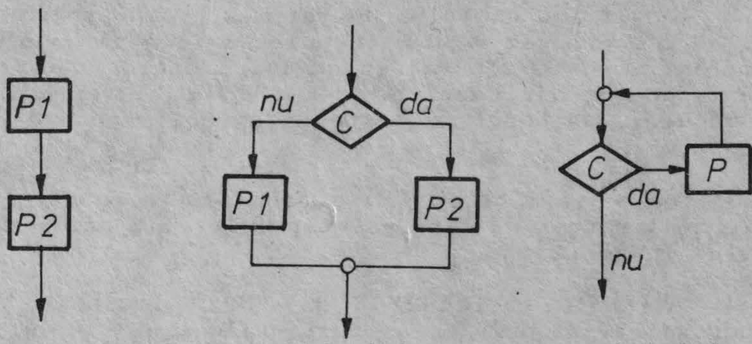
b)-modificarea unui program se poate face foarte uşor, modificarea unui modul neafectînd celelalte module;

c)-programul poate fi extins oricînd prin adăugarea de noi module;

d)-structura modulară este independentă de limbajul de programare utilizat; modulele pot fi scrise în limbaje diferite şi apoi asamblate.

Un alt principiu al programării structurate derivă din teorema de structură a lui Böhm şi Jacopini care arată că orice organigramă se poate construi folosind doar trei tipuri de structuri de control: *secvenţială* (secvenţa), *alternativă* (decizia), *repetitivă* (ciclul) prezentate în fig.1.1.

Limbajul PASCAL a fost iniţial conceput de N. Wirth pentru a înlesni însuşirea de către studenţi a principiilor "artei" programării. Noutăţile pe care le aduce acest limbaj se pot grupa în două mari categorii care sînt legate de conceperea acţiunilor şi de structurarea datelor:



a)secvenţa b)decizia c)ciclul
fig. 1.1

A)-*conceperea acţiunilor*: o serie întreagă de instrucţiuni reprezintă chiar structurile de control impuse de tehnica programării structurate (IF-THEN-ELSE, WHILE, REPEAT, FOR, CASE); acestea sînt alternativele instrucţiunii GOTO, permiţînd exprimarea în mod clar şi natural a tuturor acţiunilor posibile;

B)-*reprezentarea datelor*: s-au introdus structuri de date complexe - ca *articolul*, *mulţimea*, *fişierul*; există posibilităţi de a descrie alte structuri noi, folosind cele deja existente. De asemenea programatorul are posibilitatea să-şi definească *propriile tipuri de date*. La toate acestea se adaugă facilitatea de a defini şi manipula *structuri dinamice* (liste liniare, arbori).

Aceste avantaje ale limbajului PASCAL au făcut ca acesta să fie acceptat foarte repede de programatorii profesionişti - urmarea imediată a fost creşterea spectaculoasă a productivităţii de programare. Astfel, utilizarea limbajului PASCAL poate face scrierea programelor de 10 ori mai rapid, preţul softului scăzînd cu 30-70% (Business Week, 19 martie 1979).

Evident, odată cu apariţia şi răspîndirea fulgerătoare a calculatoarelor personale, au apărut implementări ale limbajului PASCAL pentru ele. Avînd în vedere că acest manual se adresează în primul rînd elevilor din clase de informatică şi că în general liceele cu asemenea clase au în dotare calculatoare personale **Tim-S**, **HC-85**, **COBRA** - compatibile **Spectrum**, vom descrie implementarea realizată în 1983/84 de firma **HISOFT**, care este cea mai reuşită implementare pentru acest tip de calculatoare. *Vom prezenta conceptele de bază ale limbajului PASCAL, iar acolo unde se impune vom indica specificaţiile pentru implementarea aleasă.* Menţionăm că toate programele prezentate în această carte pot fi rulate pe orice calculator personal compatibil-Spectrum fără nici o modificare.

1.2. STRUCTURA PROGRAMELOR PASCAL

Pentru a putea intui modul în care se scrie un program PASCAL, dăm pentru inceput un exemplu simplu: adunarea a două numere întregi.

```
PROGRAM ADUN;
VAR X,Y,Z:INTEGER;
BEGIN
  READ(X,Y);
  Z:=X+Y;
  WRITE(Z)
END.
```

Analizînd exemplul de mai sus, vom observa:

a) Orice program PASCAL trebuie să înceapă cu o declaraţie de program. Aceasta constă din cuvîntul cheie PROGRAM urmat de un identificator (în cazul nostru ADUN) care reprezintă numele programului. Caracterul ';' este un simbol standard şi are rol de separator.

b) Toate variabilele cu care se lucrează în program trebuie declarate, indicîndu-se tipul lor (în cazul nostru X, Y şi Z sînt întregi).

c) Cuvintele cheie BEGIN şi END delimitează corpul de instrucţiuni al blocului; instrucţiunile sînt simple: citeşte X şi Y, atribuie lui Z suma X+Y, scrie valoarea lui Z.

d) Orice program PASCAL se termină cu '.'pus după cuvîntul cheie END.

1.3. DESCRIEREA SINTAXEI LIMBAJULUI PASCAL CU AJUTORUL DIAGRAMELOR

In cele ce urmează vom descrie limbajul PASCAL şi construcţiile specifice acestuia cu ajutorul *diagramelor de sintaxă*. O diagramă de sintaxă este *un graf orientat avînd o singură intrare şi o singură ieşire - orice drum posibil de la intrare la ieşire, care urmează sensul de parcurgere indicat de săgeţi, defineşte o construcţie admisibilă sintactic*. Nodurile grafului sînt elipse, cercuri şi dreptunghiuri. Elipsele şi cercurile încadrează cuvinte cheie, respectiv simbolurile standard ale limbajului. *Acestea se regăsesc identic în textul sursă al oricărui program*. Noţiunile încadrate în dreptunghi se definesc prin alte diagrame de sintaxă (textul din dreptunghi poate fi însoţit de comentarii explicative puse între paranteze).

Utilizînd cele arătate mai sus, diagrama de sintaxă a unui program PASCAL arată astfel:

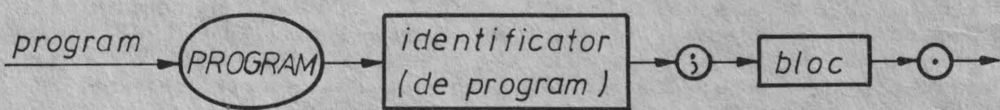


fig. 1.2.

Din diagrama din fig.1.2. se observă că unitatea de bază a programelor PASCAL este *blocul*. Acesta cuprinde descrierile datelor şi acţiunile efectuate asupra lor.

Structura generală a unui bloc este dată în fig 1.3.

Singura parte obligatorie a blocului, aşa cum se observă din diagramă, este corpul de instrucţiuni. Avînd în vedere că în PASCAL pot exista instrucţiuni vide, următorul exemplu este un program PASCAL corect:

```
PROGRAM SIMPLU;
BEGIN
END.
```

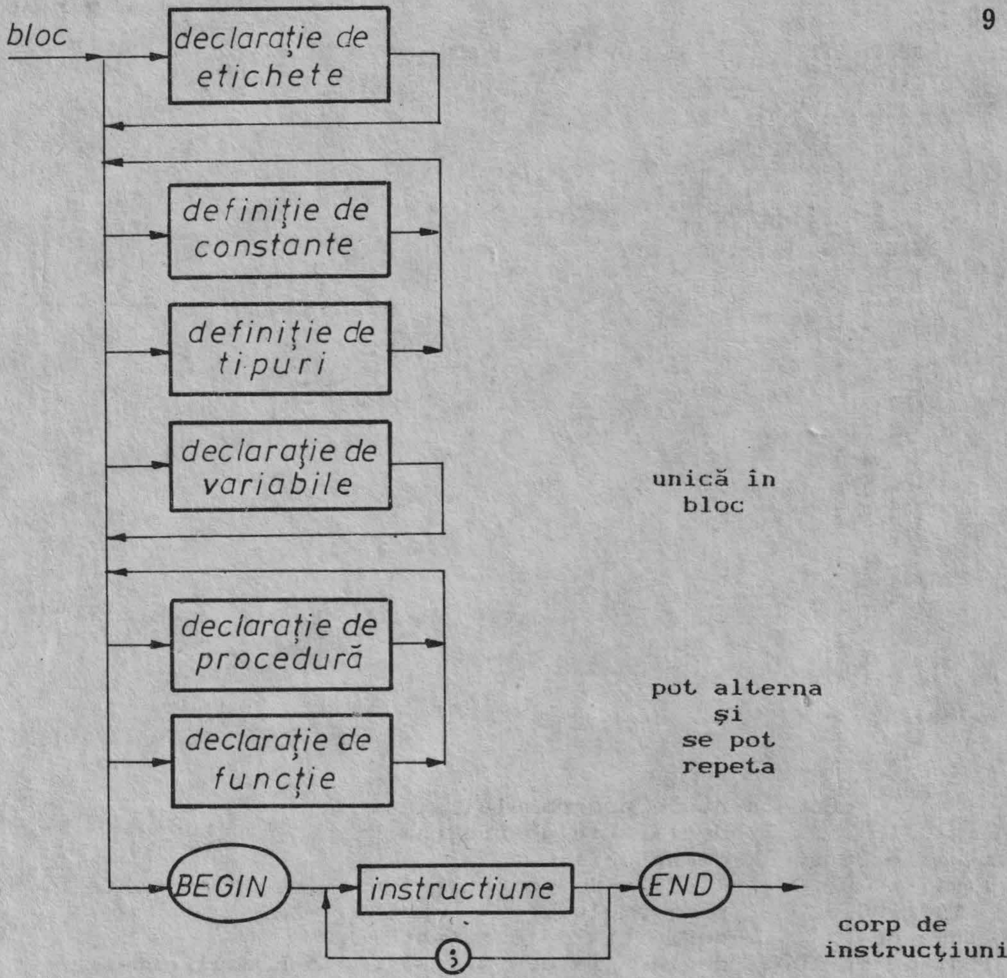


fig. I.3.

Să urmărim un program PASCAL mai complex și anume calculul tuturor permutărilor ce se pot genera cu 4 elemente.

```
AE75   10 PROGRAM PERMUTARE;
AE75   20
AE75   30 { OBTINEREA PERMUTARILOR DE N ELEMENTE FOLOSIND
AE75   40        A P E L U L   R E C U R S I V
AE75   50  se fixeaza un element si se obtin permutarile
AE75   60  pentru celelalte N-1 elemente ...          }
AE75   70
AE75   80 CONST N=4;CR=CHR(13);PR=CHR(16);
AE75   90 VAR I:INTEGER;
AE7E  100     A:ARRAY[1..N] OF 1..9;
AE7E  110
AE7E  120 PROCEDURE TIPARESTE;
AE81  130 VAR I:INTEGER;
AE81  140 BEGIN
AE99  150  FOR I:=1 TO N DO WRITE(A[I]);
AEE9  160  WRITE(CR)
AEEE  170 END;
AEF5  180
AEF5  190 PROCEDURE PERMUTA(K:INTEGER);
AEF8  200 VAR I,X:INTEGER;
AEF8  210 BEGIN
AF10  220  IF K=1 THEN TIPARESTE ELSE
AF31  230   BEGIN
AF31  240    FOR I:=1 TO K DO
AF5B  250     BEGIN
AF5E  260      X:=A[I];A[I]:=A[K];A[K]:=X;
AFF2  270      PERMUTA(K-1);
```

```
B003  280       X:=A[I];A[I]:=A[K];A[K]:=X
B08C  290      END
B097  300    END
B09B  310 END;
B0A4  320
B0A4  330 BEGIN
B0AD  340  FOR I:=1 TO N DO A[I]:=I;
B0F0  350  WRITE(PR);
B0F5  360  PERMUTA(N);
B0FE  370  WRITE(CR,PR)
B108  380 END {$P}.
```

```
2 3 4 1
3 2 4 1
3 4 2 1
4 3 2 1
2 4 3 1
4 2 3 1
4 3 1 2
3 4 1 2
3 1 4 2
1 3 4 2
4 1 3 2
1 4 3 2
2 4 1 3
4 2 1 3
4 1 2 3
1 4 2 3
2 1 4 3
1 2 4 3
2 3 1 4
3 2 1 4
3 1 2 4
1 3 2 4
2 1 3 4
1 2 3 4
```

Componentele programului sînt:

linia 10	-declaraţia de program;
liniile 30-60	-comentarii;
liniile 20,70, 110,180,320	-linii vide pentru evidenţierea componentelor programului;
linia 80	-definiţia de constante;
liniile 90-100	-declaraţia de variabile din blocul exterior;
liniile 120-170, 190-310	-două declaraţii de procedură;
liniile 330-380	-corpul de instrucţiuni al blocului exterior;

Prima procedură (liniile 120-170) se compune din:

linia 120	-numele procedurii;
linia 130	-declaraţia de variabile a procedurii;
liniile 140-170	-corpul de instrucţiuni al procedurii;

A doua procedură (liniile 190-310) se compune din:

linia 190	-numele procedurii şi parametrul de intrare K;
linia 200	-declaraţia de variabile a procedurii;
liniile 210-310	-corpul de instrucţiuni al procedurii;

Execuţia programului începe cu prima instrucţiune din corpul programului (linia 340). In acest sens se poate da o regulă generală: execuţia unui program PASCAL începe cu prima instrucţiune din blocul cel mai exterior (programul principal). Linia 360 conţine apelul procedurii PERMUTA. In acest moment este lansat în execuţie corpul de instrucţiuni al procedurii PERMUTA (liniile 210-300). După terminarea execuţiei se dezactivează procedura şi se revine la instrucţiunea care urmează apelului (linia 370). In corpul de instrucţiuni al procedurii PERMUTA se apelează şi procedura TIPARESTE, ba mai mult, se apelează chiar procedura PERMUTA! Acest mod de lucru va fi tratat la timpul potrivit (în capitolul V), urmărind aici să dăm doar o imagine generală a unui program PASCAL.

II. ELEMENTELE DE BAZA ALE LIMBAJULUI PASCAL

2.1. VOCABULARUL LIMBAJULUI PASCAL

2.1.1. Setul de caractere

Elementele constitutive ale limbajului PASCAL se construiesc pe baza următoarelor caractere:

- literele mari şi mici ale alfabetului latin, numite *caractere alfabetice*;
- cifrele sistemului de numeraţie zecimal, numite *caractere numerice*;
- simbolurile +, -, *, /, =, >, <, (,), [,], {, }, ,, ., ;, :, ', , $, numite *caractere speciale*.

2.1.2. Identificatori

Identificatorii sînt o *succesiune de litere şi cifre, primul caracter fiind obligatoriu o literă*. Identificatorii pot avea orice lungime, dar pentru unele calculatoare sînt semnificative doar primele opt caractere. Toate aceste reguli sînt redate de diagrama din fig.2.1.

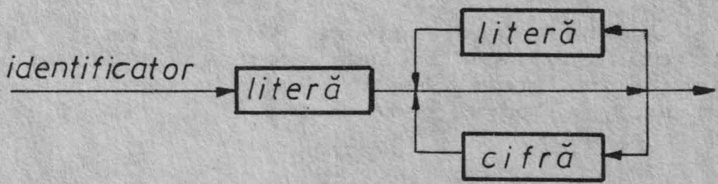


fig.2.1.

Exemple:

X, PROG1, EXEMPLU, AX13B, SUMA, A1

Următoarele nume *nu* sînt identificatori:

x-y, 7z, gr.el

Orice identificator înainte de a fi referit într-o anumită linie program trebuie să fie definit sau declarat într-o linie anterioară referirii sale.

Identificatorii desemnează nume de: programe, constante, tipuri, variabile, funcţii, proceduri, parametri.

In afara acestora există şi identificatori cu semnificaţie predefinită, cunoscuţi sub numele de *cuvinte rezervate* (*cuvinte cheie şi identificatori predeclaraţi*) ale limbajului.

Cuvintele cheie sînt predefinite pentru o anume utilizare, care va fi indicată în diagrame de sintaxă. Cuvintele cheie sînt cuprinse în tabelul T.2.1.

Tabel T.2.1.

AND	ARRAY	BEGIN	CASE	CONST	DIV
DO	DOWNTO	ELSE	END	FOR	FORWARD
FUNCTION	GOTO	IF	IN	LABEL	MOD
NIL	NOT	OF	OR	PACKED	PROCEDURE
PROGRAM	RECORD	REPEAT	SET	THEN	TO
TYPE	UNTIL	VAR	WHILE	WITH	

Observaţie:

Acestor cuvinte cheie li se mai adaugă cuvîntul FILE, dar cum structura *fişier* nu este implementată în HP4TM, el nu este trecut în tabelul de sus.

Identificatorii predeclaraţi sînt cuprinşi în tabelul T.2.2.

Tabel T.2.2.

ABS	ADDR	ARCTAN	BOOLEAN	CHAR	CHR
COS	ENTIER	EOLN	EXP	FALSE	FRAC
HALT	INCH	INLINE	INTEGER	INP	LN
MARK	MAXINT	NEW	ODD	ORD	OUT
PAGE	PEEK	POKE	PRED	RANDOM	READ
READLN	REAL	RELEASE	ROUND	SIN	SIZE
SQR	SQRT	SUCC	TAN	TIN	TOUT
TRUE	TRUNC	USER	WRITE	WRITELN	

Observaţie:

Printre identificatorii predeclaraţi în acest tabel sînt unii care sînt "cunoscuţi" numai în HP4TM; de asemenea în alte implementări utilizatorul va întîlni o serie de alţi identificatori predeclaraţi.

Aceşti identificatori sînt definiţi într-un bloc ipotetic existent care include programul utilizatorului. *Spre deosebire de cuvintele cheie, identificatorii predeclaraţi pot fi redefiniţi în cadrul programului şi utilizaţi în sensul nou dorit de programator.*

2.1.3. Separatori şi comentarii

Separatorii care delimitează două unităţi sintactice PASCAL sînt spaţiul (blancul), sfîrşitul de linie (<CR>) şi comentariul. Caracterul ';' se utilizează pentru separarea instrucţiunilor şi a declaraţiilor.

Comentariile, încadrate între delimitatorii '{' şi '}', sînt texte din care poate face parte orice caracter cu excepţia simbolurilor '{' şi '}', şi pot fi inserate oriunde în program. Sintaxa comentariilor este prezentată în fig.2.2.

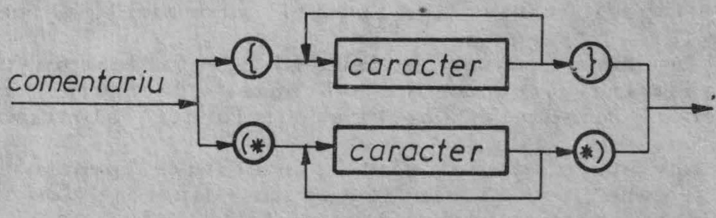


fig.2.2.

Exemple:

```
{SA STUDIEM SI PASCAL}
{REZOLVAREA ECUATIEI DE GRADUL DOI}
```

2.2. CONSTANTELE LIMBAJULUI PASCAL

Limbajul PASCAL admite următoarele constante: constante *întregi*, constante *reale*, *şiruri de caractere* şi *constante desemnate prin identificatori*. Sintaxa constantelor este prezentată în diagrama din fig.2.3.

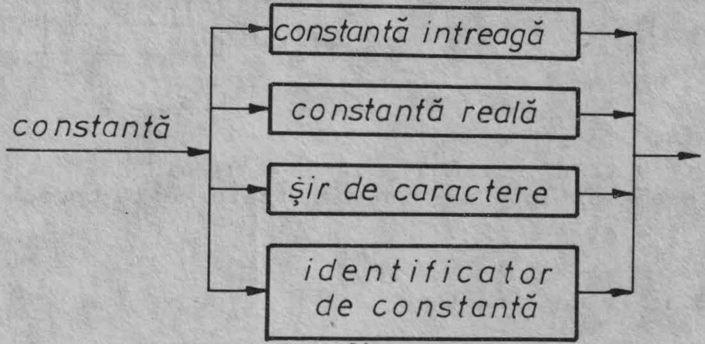


fig.2.3.

2.2.1. Constante întregi

Constantele întregi sînt numere întregi care nu pot depăşi valoarea **32767**, valoare cunoscută de compilator sub numele de **MAXINT**. Diagrama de sintaxă a constantelor întregi este cea din fig.2.4.

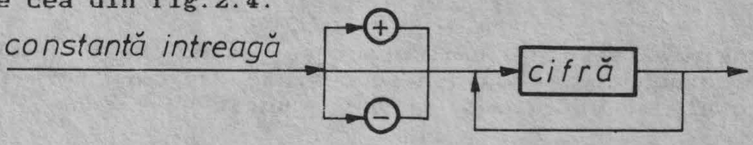


fig.2.4.

Exemple:

13, -25, 625, 1991

Următoarele numere *nu* sînt constante întregi:

76.8, 12*

MAXINT poate fi utilizat într-un program ca şi o constantă definită de programator.

Exemplu:

```
IF I<MAXINT THEN I:=I+1;
```

2.2.2. Constante reale

Constantele reale sînt numere reale cu valori absolute cuprinse între 5.9E-39 şi 3.4E38, atît partea întreagă cît şi partea zecimală trebuind să conţină cel puţin o cifră, chiar dacă această cifră este zero. Diagrama de sintaxă este dată în fig.2.5.

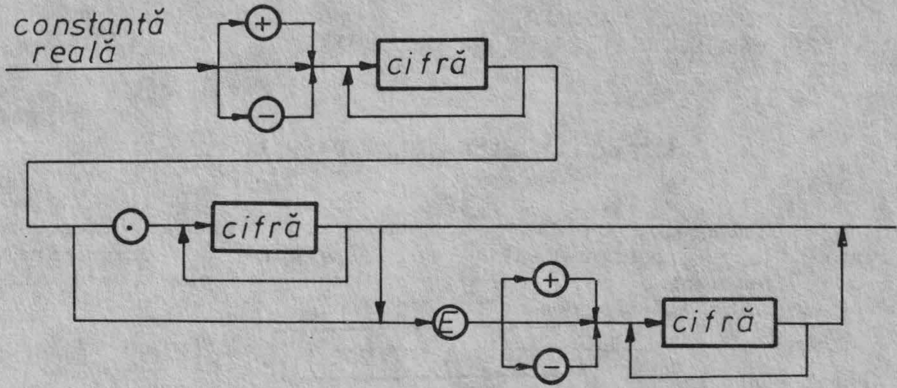


fig. 2. 5.

Exemple:

8.23145E1, 0.123E2, 1.2E-4, -42.6, 0.0, 64E-33

Următoarele constante *nu* sînt constante reale corecte:

75, 89 E-2, .77

2.2.3. Şir de caractere

O succesiune de caractere cuprinsă între apostrofuri, formează un şir de caractere. Sintaxa şirului de caractere este dată de diagrama din fig.2.6.

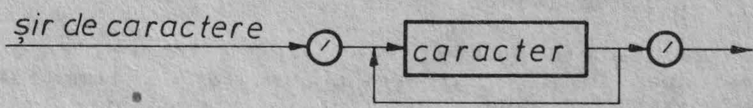


fig. 2. 6.

Lungimea şirului de caractere este chiar numărul caracterelor componente. Dacă din succesiunea de caractere face parte apostroful, atunci acesta trebuie dublat.

Exemple:

'EXEMPLU DE SIR DE CARACTERE', 'B-42 ', '30 MAI ''91'

Şirul de caractere de lungime 1, reprezintă o constantă caracter.

2.2.4. Constante desemnate prin identificatori

Identificatorii care desemnează constante în limbajul PASCAL aparţin următoarelor categorii:

- identificatorii introduşi prin definiţii de constante
- identificatorii constantelor unui tip enumerat
- identificatorii standard TRUE şi FALSE
- identificatorii NIL şi MAXINT

Toţi aceşti identificatori sînt trataţi în capitolele următoare.

2.3. DEFINIREA CONSTANTELOR UTILIZATOR

Intr-un program PASCAL, constantele pot fi utilizate direct prin valoare sau printr-un identificator. Acest identificator se defineşte în program în partea de definiţie a constantelor. Astfel, constanta este utilizată în locul valorii sale, prin identificatorul său.

Diagrama de sintaxă este cea din figura 2.7.

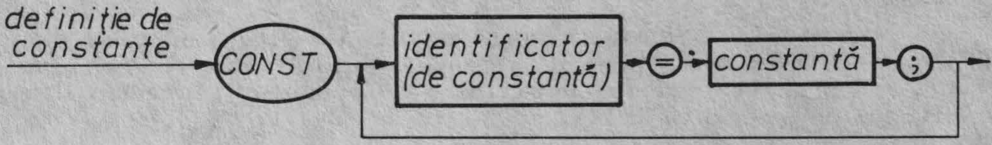


fig. 2.7.

Exemple:

```
CONST PI=3.1415926;
        E=2.7182;
        N=42;
    TITLU='PASCAL';
       BL=' ';
```

Odată stabilită valoarea identificatorului de constantă, aceasta *nu se mai poate modifica prin instrucţiuni ale programului*. Utilizarea constantelor definite prezintă o serie de avantaje:

-economie de memorie (valoarea constantei apare o dată);

-modificarea simplă a programelor (se modifică doar linia de definiţie a constantei);

-claritate sporită a programelor prin folosirea de identificatori.

Exemplu:

```
PROGRAM EXEMPLU;
CONST PI=3.1415926;
....................
  A:=PI*R*R;
  V:=(4*PI*R*R*R)/3;
```

2.4. NOŢIUNEA DE TIP

In limbajul PASCAL, orice dată - fie că reprezintă valoarea unei variabile, a unei constante sau a unei expresii - este de un anumit *tip*. *Tipul de date stabileşte mulţimea valorilor care pot fi atribuite variabilelor de tipul respectiv, cît şi mulţimea operaţiilor posibile asupra acestor valori.*

Se evidenţiază următoarele trei categorii de tipuri de date: tipuri *scalare*, tipuri *structurate* şi tipul *pointer*. Din prima categorie fac parte cele patru tipuri standard: REAL, INTEGER, BOOLEAN şi CHAR. Dintre acestea, INTEGER, BOOLEAN şi CHAR sînt tipuri *ordinale*.

Mulţimile de valori care reprezintă tipurile de date, depind de implementare.

O submulţime a numerelor întregi, care conţine constantele întregi precedate eventual de semn, formează tipul **INTEGER**.

Submulţimea finită a numerelor reale, formată din constantele reale precedate eventual de semn formează tipul **REAL**.

Mulţimea valorilor logice TRUE şi FALSE, formează tipul **BOOLEAN**.

Tipul **CHAR** este format din mulţimea caracterelor ASCII, elementele mulţimii fiind toate şirurile de caractere de lungime 1.

Tipurile INTEGER, REAL, BOOLEAN, CHAR, au fost doar trecute în revistă (pentru a putea *scrie primele programe* PASCAL), prezentarea în detaliu a fiecăruia făcîndu-se în capitolul IV şi capitolul VI.

2.5. VARIABILELE LIMBAJULUI PASCAL ŞI DECLARAREA LOR

Variabila înmagazinează valoarea unei date şi spre deosebire de o constantă, poate fi modificată pe parcursul execuţiei programului. Fiecărei variabile i se asociază un identificator, care ia valori corespunzătoare unui anumit tip.

In limbajul PASCAL, toate variabilele utilizate într-un program, trebuie declarate. Declaraţia variabilelor se face conform diagramei de sintaxă din fig 2.8.

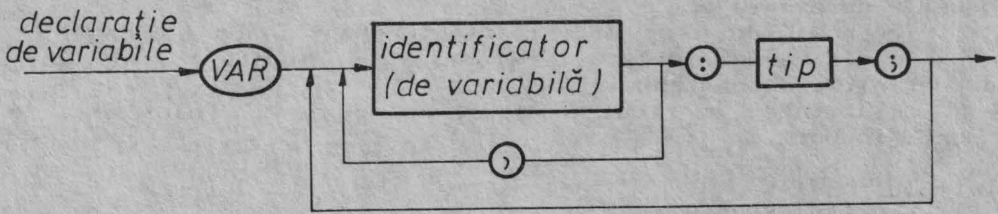


fig. 2.8.

Exemple:

a). VAR X:REAL;
A,B:INTEGER;

b). VAR NR:REAL;
A,A1:CHAR;
F:BOOLEAN;

In exemplul a) sînt declarate variabilele de tip întreg A şi B şi o varibilă X de tip real. In exemplul b) sînt declarate o variabilă reală NR, două variabile de tip caracter A, A1 şi o variabilă de tip logic F.

Noţiunea de variabilă va fi aprofundată în cadrul capitolului IV.

2.6. EXPRESII. FUNCŢII STANDARD

O succesiune de operaţii asupra unor valori în vederea obţinerii unei noi valori formează o expresie. In limbajul PASCAL, sintaxa expresiilor este dată de diagrama care urmează în fig.2.9.

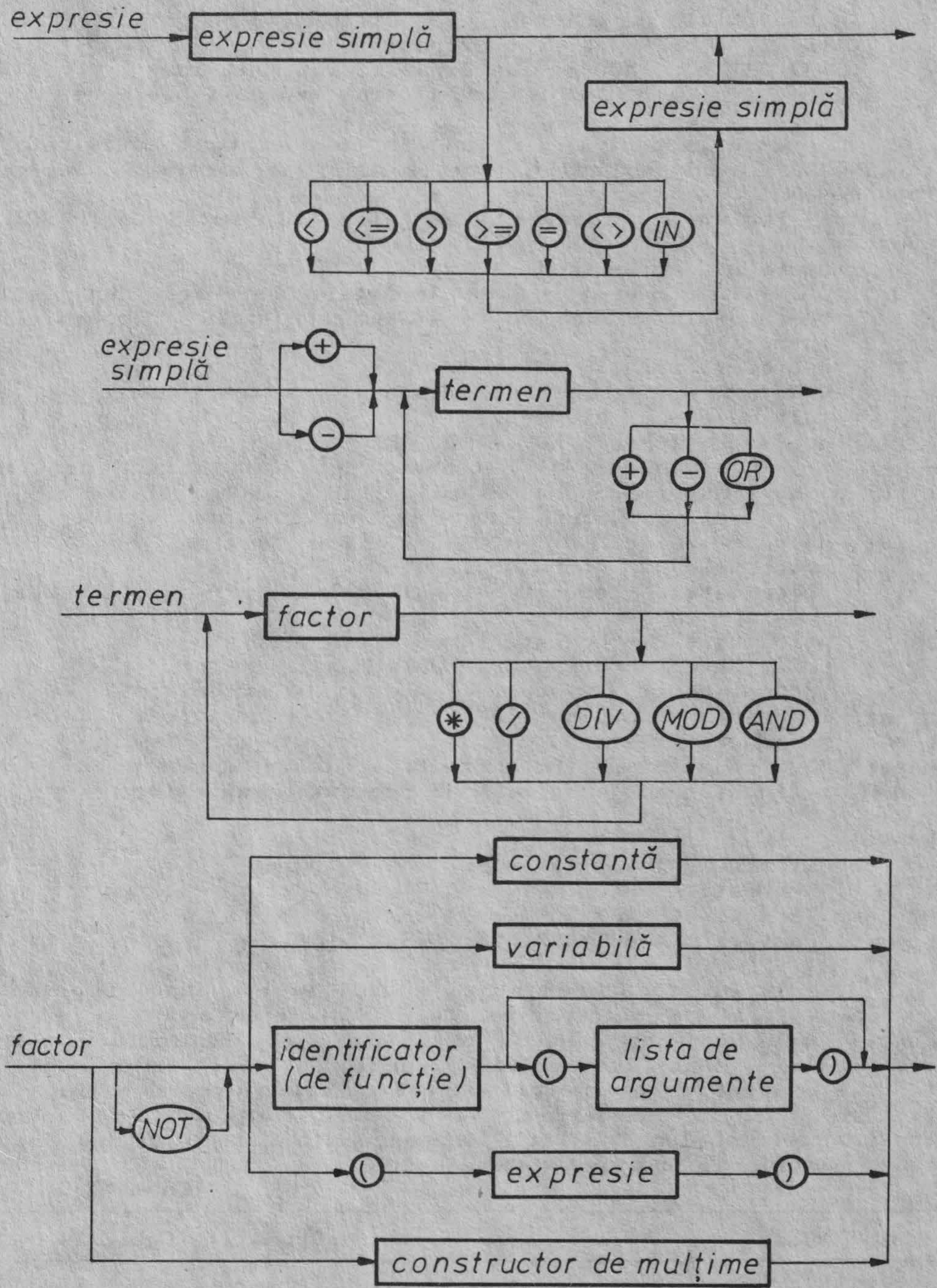


fig.2.9.

Tipul expresiei este dat de tipul valorii rezultate în urma evaluării expresiei. Acesta depinde atît de tipul operanzilor care intervin în expresia respectivă cît şi de operatorii care se aplică asupra lor.

Exemple:

```
    VAR A,B,C: REAL;
        I,J,K: INTEGER;
```

In condiţiile de mai sus, expresiile
(A+B)/2, 7.5*I, J/K sînt expresii reale,
(I+J) DIV K, J MOD 3 sînt expresii întregi. iar
A>(B+C)/7, (A+B<C) AND (J/K>A) sînt expresii booleene.

Rezultă deci că în funcţie de operatorii utilizaţi, în PASCAL avem expresii aritmetice şi expresii logice (booleene).

Pentru înţelegerea exemplelor amintim că în PASCAL avem:

- operatori aritmetici: +, -, *, /, DIV, MOD (DIV este operatorul de împărţire întreagă, MOD este operatorul pentru determinarea restului împărţirii întregi);
- operatori relaţionali: <, >, <=, >=, =, <>, IN (IN este operatorul de verificare a apartenenţei la o mulţime a unui element);
- operatori logici: NOT, AND, OR.

Operatorii vor fi trataţi detaliat, ţinînd cont de tipul de date asupra cărora se aplică, în capitolul IV.

Intre operatorii unei expresii există următoarele *reguli de prioritate*:

- prioritate 0 - operatorul NOT
- prioritate 1 - operatorii multiplicativi:*, /, DIV, MOD, AND
- prioritate 2 - operatorii aditivi:+,-,OR
- prioritate 3 - operatorii relaţionali

Operatorii cu aceeaşi prioritate se execută de la stînga la drepta în ordinea în care apar. Prezenţa parantezelor poate modifica aceste priorităţi, regula de desfacere a parantezelor fiind cea din aritmetică. Toate aceste reguli se deduc şi din diagramele de sintaxă prezentate anterior.

Exemple:

a) Expresii: A=43, X>Y, A=B;
b) Expresii simple: X-Y, A*A+B;
c) Termeni: (X<43) AND (X>Y), A/(A+B);
d) Factori: (A+B-C), X, 45, COS(X), NOT A;

In diagrama de sintaxă a unui factor apare noţiunea de *identificator de funcţie*. In PASCAL există o serie de *funcţii standard* care pot fi apelate în orice program, prin intermediul identificatorului predeclarat care este numele funcţiei, urmat de parametrii actuali. Prezentarea detaliată a fiecărei funcţii standard se va face în capitolul V. Dintre acestea, evidenţiem funcţiile standard din tabelul T.2.3., în vederea scrierii unor programe simple:

Tabel T.2.3.

FUNCŢIA	OPERAŢIA REALIZATA
ABS(X)	valoarea absolută a lui X
ARCTAN(X)	arctangentă de X
TAN(X)	tangentă de X
COS(X)	cosinus de X
SIN(X)	sinus de X
EXP(X)	"e" la puterea X
LN(X)	logaritm natural de X
SQR(X)	X la puterea 2
SQRT(X)	radical din X

Pentru funcţiile trigonometrice unghiurile se dau în radiani. Avînd în vedere că în PASCAL nu există operatori pentru ridicarea la putere şi nu a fost implementată o funcţie standard, se recomandă ca această operaţie să se realizeze pe baza formulei:

$$a^x = e^{x*\ln a}, \quad a \in R,\ a>0,\ x \in Z.$$

Exemplu:

$X = a^9$ devine în PASCAL X:=EXP(9*LN(A));

2.7. OPERAŢII DE CITIRE/SCRIERE SIMPLE

Reamintim că în această lucrare se prezintă concepte de bază PASCAL, dar detaliile se referă la implementarea HP4TM, elaborată de firma HISOFT pentru calculatoare personale compatibile Spectrum. Procedurile de intrare/ieşire prezintă multe particularităţi de la o implementare la alta. Programatorului i se recomandă, în cazul în care întîlneşte alte implementări, să lucreze cu precauţie cu procedurile de intrare/ieşire.

Citirea/scrierea datelor în PASCAL se realizează prin intermediul procedurilor **READ** şi **WRITE**, **READLN** şi **WRITELN**, a căror expunere completă se va face în capitolul V.

Printr-o procedură **READ** pot fi citite mai multe valori de tip întreg, real sau caracter, care se atribuie variabilelor desemnate de parametrii efectivi ai procedurii. Valorile citite trebuie să fie compatibile cu tipul variabilelor corespunzătoare, altfel se semnalează eroare la execuţie.

Exemplu:

```
VAR A,B:INTEGER;
    X:REAL;
    C:CHAR;
```

citire	valori introduse în timpul execuţiei de la tastatură	efect
READ(A,B);	10 20	A:=10 B:=20
READ(A,X);	10 3.5	A:=10 X:=3.5
READ(X);	13	X:=13.0
READ(A,B);	4.2 25	eroare
READ(C);	*	C:='*'
READ(A,B,C);	10 20 *	A:=10 B:=20 C:=' '
READ(A,B,C);	10 20*	A:=10 B:=20 C:='*'

Pe marginea exemplelor de mai sus se impun cîteva observaţii:

1)- valorile numerice trebuie separate;

2)- blancurile dintre valorile numerice se ignoră;

3)- avînd în vedere că blancul este şi el un caracter, atunci cînd se citesc caractere *şi acesta* va fi citit şi atribuit unei variabile de tip caracter;

4)- dacă dorim să citim o valoare de tip caracter după o valoare de tip numeric (întreg sau real) trebuie să o introducem imediat după valoarea numerică, sau de pe o linie

terminal nouă, citirea valorii numerice precedente făcîndu-se cu READLN.

5)- în cazul introducerii de la tastatură a unei valori întregi pentru citirea unei variabile reale are loc o conversie de la întreg la real, conversie permisă şi în caz de atribuire.

READLN este o procedură de citire care poate fi apelată cu sau fără parametri. READLN cu listă de parametri este echivalent cu READ cu aceeaşi listă şi un READLN fără parametri. Printr-un READLN fără parametri se ignoră restul necitit al liniei curente şi se aşteaptă citirea unei linii noi, ale cărei valori vor fi asociate cu variabilele din procedura READ sau READLN următoare.

Exemplu:

```
VAR A,B:INTEGER;
    C:CHAR;
```

citire	valori introduse de la tastatură	efect
READLN(A,B);	10 20<CR>	A:=10 B:=20
READLN(C);	*<CR>	C:='*'
READLN(A,B);	10<CR> 20<CR>	A:=10 B:=20
READLN(C);	*<CR>	C:='*'
READLN(A,B,C);	10 20<CR> *<CR>	A:=10 B:=20, *iar valoarea lui C va fi caracterul* sfîrşit de linie *al cărui cod ASCII este 13*

Din exemplele acestui paragraf rezultă:

- valorile numerice trebuie separate cu spaţiu sau <CR>;
- spaţiile şi <CR>-urile dintre valorile numerice se ignoră;
- <CR> generează un caracter (cu codul ASCII 13) care se va atribui variabilei de tip caracter imediat următoare unei variabile de tip numeric, dacă în timpul execuţiei după valoarea numerică se tastează <CR>.

Afişarea uneia sau a mai multor date de tip întreg, real, caracter sau şir de caractere, se face prin apelul procedurii **WRITE**. Valorile afişate se înscriu pe acelaşi rînd în continuare. *Valorile reale se afişează sub formă exponenţială, formă care se poate modifica utilizînd scrierea cu punct zecimal, fără exponent.*

Forma generală a scrierii cu punct zecimal, fără exponent a numerelor reale este:

WRITE(d:m:n)

unde:

d - expresie de tip real
m - număr total cifre
n - număr de cifre al părţii zecimale

Exemple:

```
WRITE(43.216:6:1);          va afişa   43.2
WRITE(43.216:4:0);          va afişa   43
WRITE(43.216:4:2);          va afişa   4.32160E+01
WRITE(-85.3);               va afişa  -.85300E+02
```

Procedura **WRITELN** realizează funcţiile procedurii WRITE, în plus, după efectuarea afişării, cursorul se mută pe linie nouă. Astfel valorile expresiilor dintr-un eventual WRITE următor se vor afişa pe linie nouă. WRITELN fără parametri are ca efect mutarea cursorului pe o linie nouă.

Exemple:

a)
```
WRITELN('A','B');
WRITELN;
WRITE('C');
WRITELN('D');
```
Secvenţa de mai sus are ca efect afişarea următoarelor rînduri:
```
linie 1    AB
linie 2
linie 3    CD
```
b)
```
WRITE('PRIMA LINIE');
WRITELN;
WRITE('LINIA A 2-A');
```
Această secvenţă are ca efect afişarea următoarelor două rînduri:
```
linie 1    PRIMA LINIE
linie 2    LINIA A 2-A
```

PROBLEME PROPUSE

1. Să se stabilească dacă următoarea secvenţă este corectă:
```
PROGRAM CORECT:
CONST NR:17;
      AD:X;
      CULOARE=ROZ;
      ORICE='ROZ';
```
şi să se corecteze în caz contrar.

2. Fie variabilele X, Y, Z, avînd valorile 2.0, 3.0 şi respectiv 4.0. Ce valori au variabilele A şi B după executarea următoarelor instrucţiuni (toate variabilele sînt de tip real):

a) X:=X+Y; Y:=Z; A:= X+Y+Z; B:=Y-X;
b) A:=SQR(X); B:=SQRT(Z)+A/X;
c) X:=SQR(X); Z:=SQRT(Z); A:=X-Z; B:=A-Y;

3. Să se scrie în PASCAL următoarele expresii:

a) $(2x-3y)^2 - 1$ b) $\sqrt{15x - 3}$

c) $\sqrt[3]{a - b} - \frac{5}{x - y}$ d) $\frac{(a - x)^2}{a + x} - \sin(x^2 + a)$

4. Fie A, B variabile de tip întreg, C de tip real şi D de tip caracter. Cum trebuie introduse valorile de la tastatură pentru următoarele citiri:

a) READ (A,B,D); pentru A=17, B=42, D='A'
b) READ (B,C); pentru B=27, C=7,2
c) READ (A,B); pentru A=8,9, B=10

5. Variabilele X şi Y sînt de tip real şi au valorile 0,5 respectiv 123,456. Ce se afişează cu instrucţiunile:

a) WRITELN(X);
b) WRITELN(X:4:1,Y:8:3);
c) WRITELN(X:6:4,Y:10:5);
d) WRITELN(X:5:2);
e) WRITE('SUMA = ',X+Y);

III. STRUCTURI FUNDAMENTALE DE CONTROL IN LIMBAJUL PASCAL

Programarea structurată urmărește elaborarea de programe pe baza unor structuri fundamentale de control în scopul obținerii unor produse program clare și fiabile. Orice program poate fi reformulat astfel încît să conțină numai structurile fundamentale de control caracteristice programării structurate: *secvențială, alternativă și repetitivă.*

Lista instrucțiunilor PASCAL este prezentată în diagrama din fig.3.1.

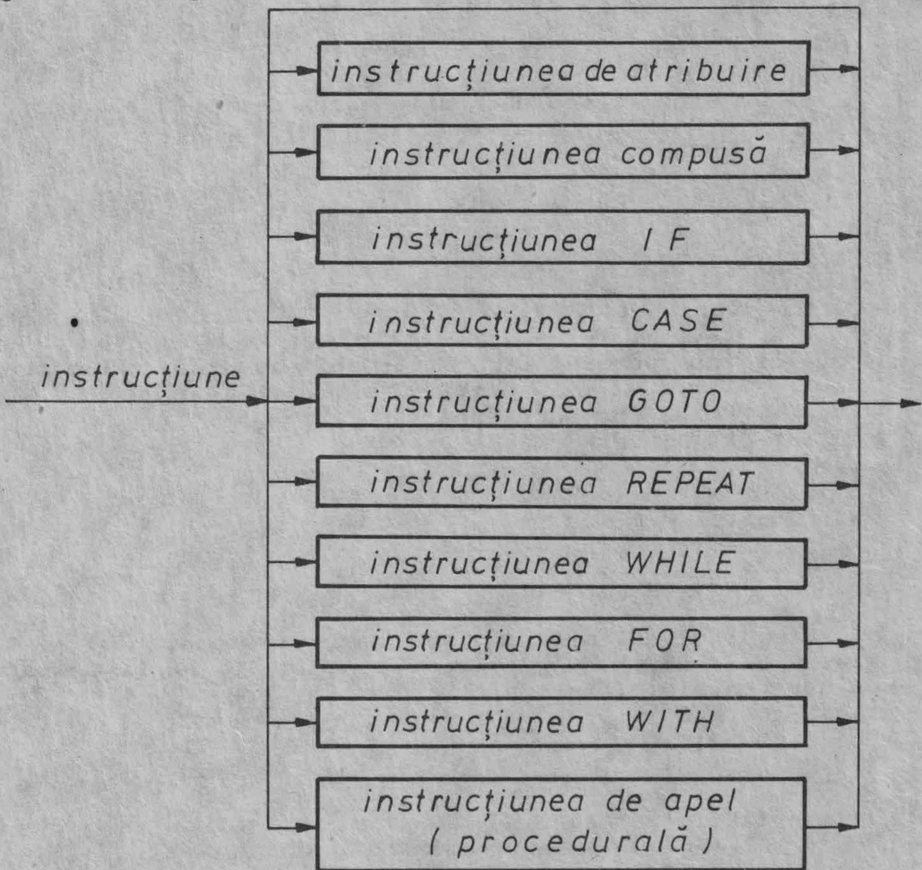


fig. 3.1.

Observație:

Dintre aceste instrucțiuni, instrucțiunea WITH va fi tra-

tată în capitolul IV, iar instrucţiunea de apel în capitolulV.

3.1. STRUCTURA SECVENŢIALA

3.1.1. Instrucţiunea de atribuire

Forma generală a structurii secvenţiale este prezentată în fig.3.2.

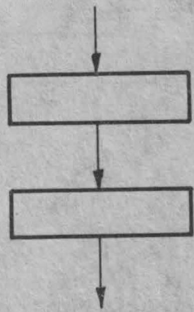


fig. 3. 2.

In limbajul PASCAL, această structură se descrie prin intermediul instrucţiunii de atribuire a cărei sintaxă este dată de diagrama din fig.3.3.

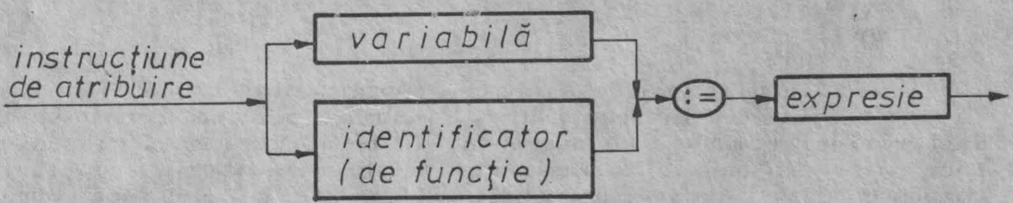


fig. 3. 3.

Efect:

1)-se evaluează expresia;
2)-se compară tipul valorii expresiei cu tipul variabilei;
-dacă tipurile coincid, are loc atribuirea;
-dacă tipurile nu coincid, atribuirea eşuează.

Observaţie:

In PASCAL, în general nu se fac conversii de tip în timpul execuţiei atribuirii. Unele implementări, printre care şi HP4TM, permit atribuirea unei valori întregi unei variabile de tip real, în prealabil efectuîndu-se conversia necesară.

Exemple:

```
VAR A,I,S:INTEGER;
    N,X:REAL;
...................
S:=7;
A:=13;
N:=S/3;
X:=(SQRT(N)+1)/I;
```

Ca urmare a efectuării instrucţiunilor de atribuire din exemplu, variabila de tip întreg A va avea valoarea 13, întregului S i se asociază valoarea 7. Lui N i se atribuie valoarea reală rezultată din împărţirea reală a întregului 7 cu 3 etc.

In PASCAL HP4TM **nu** este permisă inițializarea variabilelor la declararea lor.

3.1.2. Instrucțiunea compusă

Sintaxa instrucțiunii compuse este prezentată în fig.3.4.

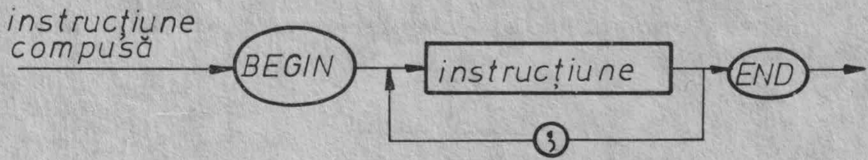


fig. 3.4.

Efectul instrucțiunii compuse se obține din înlănțuirea efectelor instrucțiunilor componente care sînt executate strict în ordinea scrierii lor. Instrucțiunea compusă *permite regruparea mai multor instrucțiuni într-una singură*. Va fi folosită ori de cîte ori sînt necesare mai multe instrucțiuni intr-o instrucțiune unde sintaxa autorizează utilizarea unei singure instrucțiuni.

Exemplu:
```
BEGIN
  X1:=k+1;
  X2:=(A1+A2)/2;
END;
```

Din diagrama de sintaxă a programului PASCAL rezultă că și corpul de instrucțiuni al unui bloc are forma unei instrucțiuni compuse. De asemenea, instrucțiunea compusă poate face parte din ansamblul instrucțiunilor unei alte instrucțiuni compuse, caz în care perechile BEGIN-END se grupează după regula din matematică a parantezelor. Prezența separatorului ';' este obligatorie între instrucțiuni și opțională înainte de END. Dacă se pune ';' înainte de END, atunci între ';'și END se consideră o *intrucțiune vidă*. Efectul instrucțiunii vide este nul.

PROGRAMUL III.1.

Să se calculeze volumul unui con circular drept cu generatoarea și raza bazei date.

```
algoritmul VOL este:
  citeste r,g
  pi ← 3,1415926
  h ← √(g²-r²)
  v ← pi*r²*h/3
  scrie v
stop
```

Programul PASCAL corespunzător este programul **VOL** (P.III.1). In linia 20 este definită constanta PI, iar în liniile 30-40 sînt declarate variabilele utilizate. Deoarece funcția standard SQRT are valoare reală, variabila H, care identifică înălțimea, se declară ca o variabilă de tip real. De asemenea variabila V (volumul) este declarată tot de tip real, deoarece operatorul aritmetic '/' produce rezultat real. Corpul de instrucțiuni este format din liniile 60-120 care realizează

citirea razei şi a generatoarei conului, calculul înălţimii şi a volumului, cît şi afişarea valorii calculate pentru volum. La afişare s-a folosit reprezentarea cu punct zecimal, fără exponent a variabilei reale V pe 10 poziţii, din care 2 zecimale.

In vederea utilizării imprimantei, în program apare instrucţiunea WRITELN(CHR(16)), care asigură comutarea ieşirii de la monitor la imprimantă şi viceversa. In acelaşi scop, în linia 130 apare şi opţiunea de compilare P care comută canalul de ieşire de la monitor la imprimantă.

```
ACEC  10 PROGRAM VOL;
ACEC  20 CONST PI=3.1415926;
ACEC  30 VAR R,G:INTEGER;
ACF5  40     V,H:REAL;
ACF5  50 BEGIN
ACFE  60   READ(R,G);
AD0A  70   H:=SQRT(G*G-R*R);
AD3F  80   V:=PI*SQR(R)*H/3;
AD71  90   WRITELN(CHR(13));
AD7B   100   WRITELN('RAZA:',R,'    GENERATOAREA:',G);
ADBC   110   WRITELN('VOLUMUL CONULUI:',V:10:2);
ADEE   120   WRITELN(CHR(13))
ADF5   130 END {$P}.


RAZA:5      GENERATOAREA:8
VOLUMUL CONULUI:    163.49
```

P.III.1.

PROGRAMUL III.2.

Dîndu-se lungimile laturilor unui triunghi,să se calculeze aria triunghiului şi lungimile înălţimilor sale.

```
┌ algoritmul ARIA1 este:
│   citeste a,b,c
│   p  ← (a+b+c)/2
│   s  ← √(p*(p-a)*(p-b)*(p-c))
│   ha ← 2*s/a
│   hb ← 2*s/b
│   hc ← 2*s/c
│   scrie s,ha,hb,hc
└ stop
```

Implementarea în PASCAL este dată în programul **ARIA1** (P.III.2).

```
AD75   10 PROGRAM ARIA1;
AD75   20 VAR A,B,C:INTEGER;
AD7E   30      P,S,HA,HB,HC:REAL;
AD7E   40 BEGIN
AD87   50   READ(A,B,C);
AD99   60   WRITELN(CHR(13),'A=',A);
ADB9   70   WRITELN('B=',B);
ADD2   80   WRITELN('C=',C);
ADEB   90   P:=(A+B+C)/2;
AE19  100   S:=SQRT((P*(P-A)*(P-B)*(P-C)));
AE7B  110   HA:=2*S/A;
AE9F  120   HB:=2*S/B;
AEC3  130   HC:=2*S/C;
AEE7  140   WRITELN('ARIA=',S:10:2);
AF0E  150   WRITELN('INALTIMEA HA=',HA:10:2);
AF3D  160   WRITELN('INALTIMEA HB=',HB:10:2);
AF6C  170   WRITELN('INALTIMEA HC=',HC:10:2,CHR(13))
AF9F  190 END {$P}.
```

```
A=30
B=40
C=50
ARIA=    600.00
INALTIMEA HA=    40.00
INALTIMEA HB=    30.00
INALTIMEA HC=    24.00
```

P.III.2.

3.2. STRUCTURA ALTERNATIVA

Forma generală a structurii alternative, numită şi IF-THEN-ELSE, este prezentată în fig.3.5.

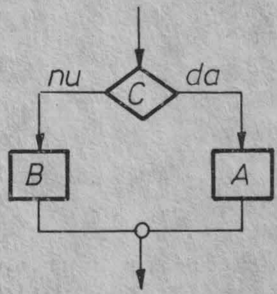


fig. 3. 5.

Dacă A sau B lipseşte, structura se numeşte structură alternativă simplă sau pseudoalternativă (IF-THEN).

3.2.1. Instrucţiunea IF

In limbajul PASCAL, structura alternativă se descrie cu ajutorul instrucţiunii IF, a cărei sintaxă este prezentată în fig.3.6.

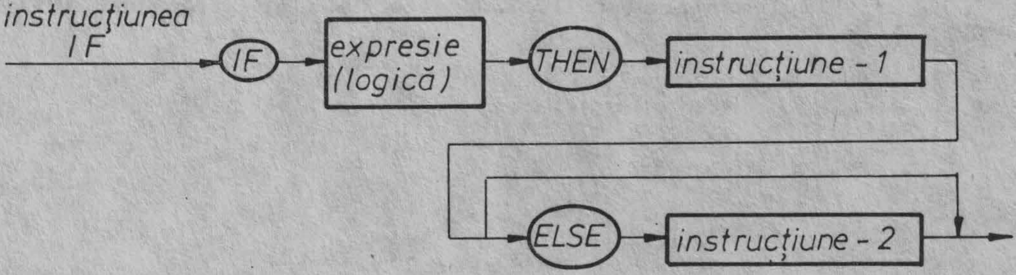


fig. 3. 6.

Efect:

1)-se evaluează expresia logică;

2)-dacă valoarea logică a expresiei este TRUE, se execută instrucţiune-1;

3)-dacă valoarea logică a expresiei este FALSE, ori se execută instrucţiune-2 (dacă este prezentă alternativa ELSE), ori nu se execută nimic;

4)-se părăseşte instrucţiunea IF.

Instrucţiune-1 şi instrucţiune-2 pot fi orice instrucţiune PASCAL, inclusiv instrucţiunea vidă.

Exemple:

```
a)   IF X=0 THEN S:=S+I
            ELSE P:=P*I;

b)   IF A>B THEN
              BEGIN
                X:=A+B;
                Y:=A;
                Z:=B;
                WRITE(X,Y,Z)
              END
            ELSE WRITE(A,B);

c)   IF (X>A) AND (X<B) THEN WRITE(X);

d)   IF X=0 THEN
            ELSE X:=X+1;

e)   IF Y>10 THEN Y:=Y-1
             ELSE;
```

Observaţii:

1)-*înaintea cuvîntului cheie ELSE nu se pune ';'*

2)-dacă oricare ramură a instrucţiunii IF conţine mai multe instrucţiuni atunci acestea se grupează într-o singură *instrucţiune compusă*;

3) atît instrucţiune-1 cît şi instrucţiune-2 pot conţine alte instrucţiuni IF, caz în care *asocierea unui ELSE se face cu THEN-ul cel mai apropiat care îl precede şi care nu a fost asociat încă.* Dacă se impune o altă asociere, pentru evitarea oricărei ambiguităţi se recomandă *soluţiile* ilustrate în exemplul următor.

Exemplu:

Fie următoarea structură:

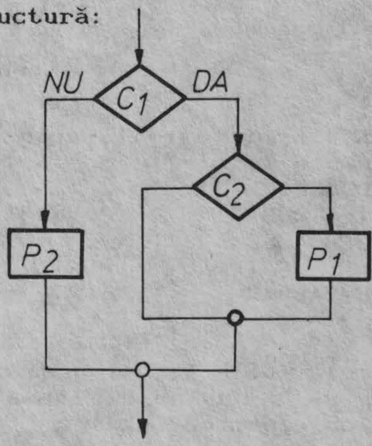


fig.3.7.

```
1)        if c1                2)        if c1
            then if c2                     then
                   then p1                   begin
                   else                         if c2
            else p2;                              then p1
                                               end
                                             else p2;
```

Soluţia 1) utilizează *else* vid, astfel evitînd ataşarea ramurii *else* *p*2 cazului *c*2=*false*.

Soluţia 2) delimitează instrucţiunea IF (care trebuie executată cînd *c*1=*true*) într-o instrucţiune compusă, chiar dacă acest lucru nu se impune de sintaxă deoarece avem o singură instrucţiune.

4) O instrucţiune IF formulată neglijent poate fi foarte costisitoare. Să considerăm *n* condiţii reciproc independente $c_1, c_2, \ldots, c_n$ care provoacă, fiecare, execuţia cîte unei secvenţe.

Fie condiţiile:

```
if c1 then s1;
if c2 then s2;
.............
if cn-1 then sn-1;
if cn then sn;
```

Pentru a grăbi ieşirea din IF, vom aşeza condiţiile în ordinea descrescătoare a probabilităţii îndeplinirii lor.

In cazul în care din condiţiile de mai sus *numai una poate fi îndeplinită la un moment dat*, se recomandă următorul model:

```
if c1
  then s1
  else if c2
         then s2
         else.....
          ...........
                 else if cn-1
                        then sn-1
                        else if cn
                               then sn;
```

5)-Pentru *a evita utilizarea abuzivă* a instrucţiunii IF se recomandă ca în loc de

```
IF CHEIE=VALOARE
  THEN GASIT:=TRUE
  ELSE GASIT:=FALSE;
```

să se scrie:

```
GASIT:=CHEIE=VALOARE;
```

PROGRAMUL III.3.

Se dau numerele reale X,Y şi numărul intreg N. Dacă N este diferit de zero, să se calculeze suma numerelor X,Y, în caz contrar să se calculeze produsul lor.

```
┌─ algoritmul PIII3 este:
│    citeste x,y,n
│  ┌─daca n=0 atunci p ←— x*y
│  │                 scrie p
│  │          altfel s ←— x+y
│  │                 scrie s
│  └─sfdaca
└─ stop
```

Programul PASCAL corespunzător este P1 (P.III.3), el exemplifică structura alternativă completă.

```
AD05   10 PROGRAM P1;
AD05   20 VAR X,Y,P,S:REAL;
AD0E   30     N:INTEGER;
AD0E   40 BEGIN
AD17   45   READ(N,X,Y);
AD31   50   WRITELN;
AD34   60   WRITELN('N=',N);
AD4D   70   WRITELN('X=',X:5:2);
AD71   80   WRITELN('Y=',Y:5:2);
AD95   90   IF N=0 THEN
ADA7  100          BEGIN
ADA7  110            P:=X*Y;
ADC1  120            WRITELN('PRODUSUL=',P:10:2)
ADE9  130          END
ADEC  140        ELSE
ADEF  150          BEGIN
ADEF  160            S:=X+Y;
AE09  170            WRITELN('SUMA=',S:10:2)
AE2D  180          END;
AE30  190   WRITELN
AE30  200 END {$P}.
```

```
N=0
X=45.78
Y=13.13
PRODUSUL=    601.09

N=72
X=53.12
Y=75.73
SUMA=    128.85
```

P.III.3.

PROGRAMUL III.4.

Să se calculeze maximul a trei numere reale date.

```
┌ algoritmul MAX este:
│   citeste a,b,c
│   m ←— a
│   daca m<b atunci m ←— b sfdaca
│   daca m<c atunci m ←— c sfdaca
│   scrie m
└ stop
```

Transcrierea în limbaj PASCAL este dată în programul **MAX** (P.III.4) care ilustrează structura pseudoalternativă ; în exemplul dat, ramura corespunzătoare neindeplinirii condiţiei este vidă.

```
ACD7  10 PROGRAM MAX;
ACD7  20 VAR A,B,C,M:REAL;
ACE0  30 BEGIN
ACE9  40   READ(A,B,C);
AD07  50   M:=A;
AD15  60   IF M<B THEN M:=B;
AD42  70   IF M<C THEN M:=C;
AD6F  80   WRITELN(CHR(13));
AD79  90   WRITELN('A=',A:6:2,'  ','B=',B:6:2,'  ','C=',C:6:2);
ADF9   100   WRITELN('MAXIMUL ESTE:',M:6:2);
AE28   110   WRITELN(CHR(13))
AE2F   120 END {$P}.
```

```
A= 23.34  B= 40.00  C=  0.76
MAXIMUL ESTE: 40.00


A= 34.00  B=789.65  C= 54.00
MAXIMUL ESTE:789.65
```

P.III.4.

PROGRAMUL III.5.

Să se calculeze valoarea funcţiei reale de variabilă reală, definită prin formula:

$$f(x)=\begin{cases} x-3 & \text{dacă } x<=0 \\ 2-x & \text{dacă } 0<x<=2 \\ -x^2 & \text{dacă } x>2 \end{cases}$$

Valoarea variabilei x se citeşte.

```
algoritmul FUNCTIE este:
   citeste x
   daca x<=0 atunci f ← x-3
             altfel daca x<=2 atunci f ← 2-x
                              altfel f ← -x²
                    sfdaca
   sfdaca
   scrie f
stop
```

Implementarea acestui algoritm în PASCAL este dată de programul **FUNCTIE** (P.III.5). In acest program se poate observa modul de asociere a unui ELSE cu THEN-ul cel mai apropiat care-l precede în cazul unui IF inclus în IF.

```
AE1B   10 PROGRAM FUNCTIE;
AE1B   20 VAR X,F:REAL;
AE24   30 BEGIN
AE2D   40  WRITE('X=');READ(X);
AE44   50  WRITELN('X=',X:6:2);
AE68   60  IF X<=0 THEN F:=X-3
AEA1   70         ELSE
AEB0   80            IF X<=2 THEN F:=2-X
AEE0   90                   ELSE F:=-X*X;
AF1A  100  WRITELN('VALOAREA FUNCTIEI ESTE:');
AF3F  110  WRITELN('F(',X:6:2,')=',F:10:4);
AF84  120 END {$P}.


X= 10.00
VALOAREA FUNCTIEI ESTE:
F( 10.00)= -100.0000

X= -3.00
VALOAREA FUNCTIEI ESTE:
F( -3.00)=   -6.0000

X= 45.30
VALOAREA FUNCTIEI ESTE:
F( 45.30)=-2052.0887
```

P.III.5.

3.2.2. Instrucţiunea CASE

Instrucţiunea CASE este o generalizare a instrucţiunii IF în cazul selecţiei pentru mai mult de două alternative posibile. Sintaxa instrucţiunii este dată de diagrama din fig.3.8.

Expresia din sintaxa instrucţiunii CASE se numeşte *selector*, iar constantele se numesc *constante CASE*. Selectorul şi constantele CASE trebuie să fie de acelaşi tip *ordinal*

Efect:

1)-se evaluează selectorul;

2)-se caută alternativa CASE în a cărei listă de constante CASE se regăseşte valoarea selectorului;

3)-în cazul în care se găseşte o asemenea alternativă, se execută instrucţiunea respectivă;

4)-în caz contrar se execută instrucţiunea specificată în ELSE (dacă există);

5)-se iese din instrucţiunea CASE.

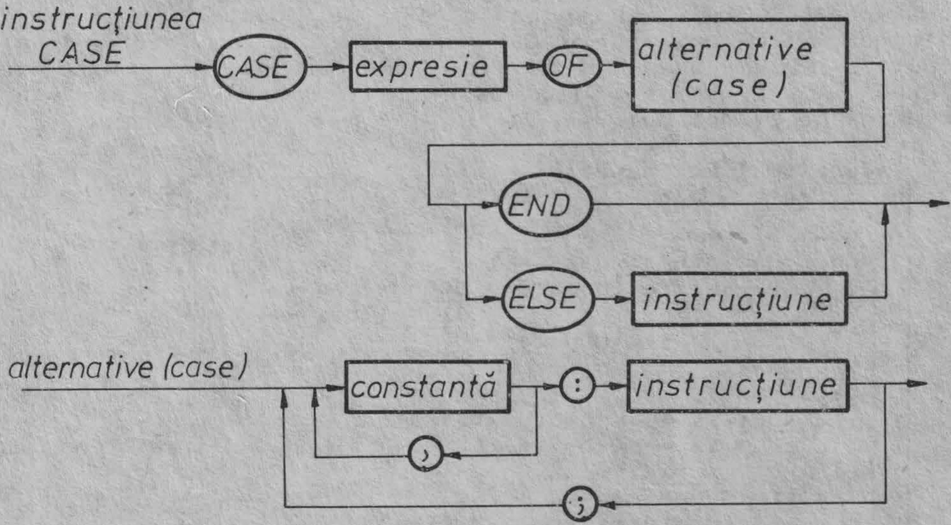


fig. 3. 8.

Exemple:

a) VAR A:INTEGER;

```
   ..............
     CASE A OF
            0:X:=100;
          1,7:X:=200;
        5,3,4:X:=300
     END;
```

b) VAR ZI:CHAR;

```
   ..............
     CASE ZI OF
       'L':WRITE('LUNI');
       'M':WRITE('MARTI SAU MIERCURI');
       'J':WRITE('JOI');
       'V':WRITE('VINERI')
         ELSE WRITE('ZI DE ODIHNA')
     END;
```

Observaţii:

1)-constantele CASE trebuie să fie *distincte* între ele, dar nu trebuie să fie puse în ordine;

2)-pot fi *cel mult 255* constante CASE;

3)-alternativele CASE *nu pot fi etichetate*, deci nu pot fi referite cu o instrucţiune GOTO;

4)-implementare instrucţiunii CASE în HP4TM nu permite caracterul ';' în faţa alternativei ELSE şi nici în faţa cuvîntului cheie END;

5)-nu toate implementările permit forma cu ELSE a instrucţiunii CASE.

PROGRAMUL III.6.

Să se determine soluţiile reale ale ecuaţiei de gradul doi cu coeficienţi numere reale, nenule.

Programul EC2 (P.III.6) conţine în liniile 20-30 un comentariu. In liniile 120-150 calculează discriminantul D al ecuaţiei şi după cum acesta este pozitiv, zero sau negativ, variabila CAZ ia respectiv valorile 1, 2 sau 3. Variabila CAZ este selectorul instrucţiunii CASE, iar 1, 2 şi 3 constituie constantele CASE corespunzătoare. Se poate observa că selectorul şi constantele CASE sînt de acelaşi tip ordinal, tipul INTEGER. Programul a fost executat de mai multe ori pentru a fi ilustrate toate cazurile prevăzute în rezolvarea ecuaţiei de gradul doi.

```
┌algoritmul EC2 este:
│  citeste a,b,c
│  d ← b²-4*a*c
│ ┌daca d>0 atunci caz ← 1
│ │        altfel┌daca d=0 atunci caz ← 2
│ │              │           altfel caz ← 3
│ │              └sfdaca
│ └sfdaca
│ ┌daca caz=1
│ │  atunci x1 ← (-b+ √d)/(2*a)
│ │         x2 ← (-b- √d)/(2*a)
│ │         scrie x1,x2
│ │  altfel┌daca caz=2
│ │        │  atunci x ← -b/(2*a)
│ │        │         scrie x
│ │        │  altfel┌daca caz=3
│ │        │        │  atunci scrie 'nici o soluţie reală'
│ │        │        └sfdaca
│ │        └sfdaca
│ └sfdaca
└stop
```

```
AFD7   10 PROGRAM EC2:
AFD7   20 (APLICATIE LA CASE-OF:
AFD7   30                REZOLVAREA ECUATIEI DE GRADUL DOI)
AFD7   40 VAR A,B,C,D,X,X1,X2:REAL;
AFE0   50                CAZ:INTEGER;
AFE0   60 BEGIN
AFE9   70   WRITELN;WRITELN('INTRODU A,B,C');WRITELN:
B00A   80   READ(A,B,C);
B028   90   WRITELN;
B02B  100   WRITELN('A=',A:6:2,'    ','B=',B:6:2,'    ','C=',C:6:2);
B0AD  110   WRITELN;
B0B0  120   D:=B*B-4*A*C;
B0F1  130   IF D>0 THEN CAZ:=1
B11B  140         ELSE IF D=0 THEN CAZ:=2
B145  150                    ELSE CAZ:=3;
B150  160   CASE CAZ OF
B153  170     1:BEGIN
B15D  180         X1:=(-B+SQRT(D))/(2*A);
B199  190         X2:=(-B-SQRT(D))/(2*A);
B1D9  200         WRITELN('DOUA SOLUTII:');WRITELN;
B1F7  210         WRITELN('X1=',X1:6:2);WRITELN;
B21F  220         WRITELN('X2=',X2:6:2);
B244  230       END;
B247  240     2:BEGIN
B251  250         X:=-B/(2*A);
B27E  260         WRITELN('O SOLUTIE');WRITELN;
B298  270         WRITELN('X=',X:6:2);
B2BC  280       END;
B2BF  290     3:WRITELN('NICI O SOLUTIE')
B2E2  300   END;
B2E5  310   WRITELN
B2E5  320 END ($P).
```

```
A=  1.20   B=  1.00   C=  1.00

NICI O SOLUTIE
```

```
A=  1.00   B= -2.00   C=  1.00

O SOLUTIE

X=  1.00


A=  1.00   B= -5.00   C=  6.00

DOUA SOLUTII:

X1=  3.00

X2=  2.00
```

P.III.6

PROGRAMUL III.7.

Fiind date numerele reale A, B şi N natural, să se calculeze valoarea E astfel:

$$E=\begin{cases} A+B & \text{dacă } N=1 \\ A-B & \text{dacă } N=2 \\ A*B & \text{dacă } N=3 \text{ sau } N=4 \\ (A+B)^2 & \text{dacă } N=5 \end{cases}$$

```
algoritmul EXPRESIE este:
  citeste n,a,b
  daca n=1
    atunci e <- a+b
           scrie e
    altfel daca n=2
             atunci e <- a-b
                    scrie e
             altfel daca n=4 V n=3
                      atunci e <- a*b
                             scrie e
                      altfel daca n=5
                               atunci e <- (a+b)^2
                                      scrie e
                               altfel scrie 'n eronat'
                             sfdaca
                    sfdaca
           sfdaca
  sfdaca
stop
```

Avînd în vedere că în acest algoritm structurile alternative ramifică execuţia în funcţie de valoarea unei variabile de tip ordinal, în programul **EXPRESIE** (P.III.7) vom lucra cu instrucţiunea CASE, astfel mărind lizibilitatea programului. Alternativa ELSE din CASE corespunde tuturor cazurilor cînd N nu este din intervalul [1,5].

```
AD67    PROGRAM EXPRESIE;
AD67    VAR A,B,E:REAL;
AD70        N:INTEGER;
AD70    BEGIN
AD79      READ(N,A,B);
AD93      WRITELN('N=',N);
ADAC      WRITELN('A=',A:8:2);
ADD0      WRITELN('B=',B:8:2);
ADF4         CASE N OF
ADF7            1:BEGIN
AE01                E:=A+B;
AE1B                WRITELN('E=',E:10:2)
AE3C              END;
```

```
AE42          2:BEGIN
AE4C              E:=A-B;
AE6A              WRITELN('E=',E:10:2)
AE8B            END;
AE91          4,3:BEGIN
AEA5                E:=A*B;
AEBF                WRITELN('E=',E:10:2)
AEE0              END;
AEE6          5:BEGIN
AEF0              E:=SQR(A+B);
AF0D              WRITELN('E=',E:10:2)
AF2E            END
AF31           ELSE WRITELN('N ERONAT');
AF4A      END (*P).
```

```
N=1
A=    4.60
B=    7.00
E=     11.60


N=2
A=    4.00
B=    6.20
E=     -2.20


N=4
A=    6.00
B=    2.00
E=     12.00


N=5
A=    7.76
B=    8.45
E=    262.76


N=11
A=    1.00
B=   -7.00
N ERONAT
```

P.III.7.

3.3. STRUCTURA REPETITIVA

Structura repetitivă este descrisă cu ajutorul instrucţiunilor repetitive, care permit repetarea în anumite condiţii a unei instrucţiuni sau a unei secvenţe de instrucţiuni.

Instrucţiunea repetată formează *corpul ciclului*, ciclu care constituie instrucţiunea repetitivă.

Cele trei tipuri ale structurii repetitive se descriu în PASCAL cu ajutorul următoarelor instrucţiuni:

1)-instrucţiunea WHILE (numită *ciclu cu test iniţial*, sau ciclu anterior condiţionat) descrie structura de tipul WHILE-DO;

2)-instrucţiunea REPEAT (numită *ciclu cu test final* sau ciclu posterior condiţionat) descrie structura de tipul DO-UNTIL;

3)-instrucţiunea FOR (numită *ciclu cu contor* sau ciclu cu număr de repetiţii cunoscut) descrie structura de tipul DO-FOR.

3.3.1. Instrucțiunea WHILE

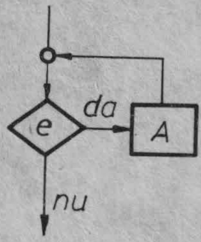


fig. 3.9.

Structura repetitivă de tipul WHILE-DO are forma generală din fig.3.9.

In limbajul PASCAL, această structură se descrie cu instrucțiunea WHILE, a cărei sintaxă este dată de diagrama din fig.3.10.

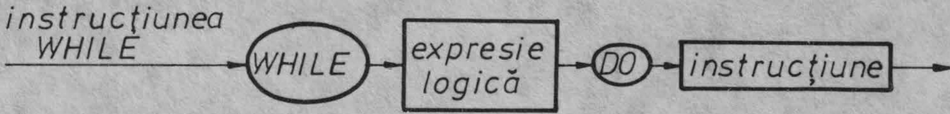


fig. 3.10.

Efect:

1)-se evaluează expresia logică;

2)-*cît timp* expresia logică are valoarea adevărat (TRUE) se execută corpul ciclului (instrucțiunea specificată după cuvîntul cheie DO). Dacă corpul ciclului se compune din mai multe instrucțiuni atunci acestea se grupează într-o instrucțiune compusă.

Exemple:

a)
```
X:=0;
WHILE X<=N DO
  X:=X+1;
```

b)
```
I:=1;P:=1;
WHILE I<=N DO
  BEGIN
    P:=P*I;
    I:=I+1
  END;
```

Observații:

1)- dacă condiția de ciclare nu este îndeplinită la intrarea în ciclu atunci *instrucțiune* nu se va executa niciodată;

2)- deoarece expresia logică se evaluează la fiecare iterație, ea trebuie să fie cît se poate de simplă.

PROGRAMUL III.8.

Pentru primele N numere naturale să se listeze o tabelă de forma N, N^2, N^3, $\sqrt{N}$, unde N este dat.

<u>algoritmul</u> LIST <u>este</u>:
  <u>citeste</u> n
  i ← 1
  <u>cittimp</u> i<=n <u>executa</u>
    <u>scrie</u> i, i^2, i^3, $\sqrt{i}$
    i ← i+1
  <u>sfcittimp</u>
<u>stop</u>

Programul **LIST** (P.III.8) realizează tabelarea cerută pe patru coloane.

```
ADF1   10 PROGRAM LIST;
ADF1   20 VAR I,N:INTEGER;
ADFA   30 BEGIN
AE03   40   READ(N);
AE09   50   WRITELN(CHR(13));
AE13   55   I:=1;
AE19   60   WHILE I<=N DO
AE31   70   BEGIN
AE31   80       WRITELN(I:2,I*I:5,I*I*I:7,SQRT(I):8:5);
AE86   90       I:=I+1
AE8E  100   END;
AE90  110   WRITELN(CHR(13))
AE97  120 END {$P}.
```

```
 1    1     1 1.00000
 2    4     8 1.41421
 3    9    27 1.73205
 4   16    64 2.00000
 5   25   125 2.23607
 6   36   216 2.44949
 7   49   343 2.64575
 8   64   512 2.82843
 9   81   729 3.00000
10  100  1000 3.16228
11  121  1331 3.31662
12  144  1728 3.46410
13  169  2197 3.60555
14  196  2744 3.74166
15  225  3375 3.87298
16  256  4096 4.00000
17  289  4913 4.12311
18  324  5832 4.24264
19  361  6859 4.35890
20  400  8000 4.47214
```

P.III.8.

PROGRAMUL.III.9.

Să se efectueze împărţirea a două numere naturale prin scăderi succesive.

```
┌ algoritmul CIT este:
│   citeste a,b
│   c ← 0
│ ┌ cittimp a>=b executa
│ │   d ← a-b
│ │   c ← c+1
│ │   a ← d
│ └ sfcittimp
│   scrie c,a
└ stop
```

Algoritmul este implementat în programul CIT (P.III.9). Corpul ciclului din P.III.9. este o instrucţiune compusă, descris în liniile 70-110. Numerele care se împart sînt afişate pe prima linie, cîtul şi restul împărţirii pe o a doua linie.

```
AE2C   10 PROGRAM CIT;
AE2C   20 VAR A,B,C,D:INTEGER;
AE35   30 BEGIN
AE3E   40   READ(A,B);
AE4A   50   WRITELN('A=',A,'      ','B=',B);
AE88   60   C:=0;
AE8E   70   WHILE A>=B DO
AEA7   80        BEGIN
AEA7   90            D:=A-B;
AEB9  100            C:=C+1;
AEC0  110            A:=D
AEC0  120            END;
AEC9  130   WRITELN(CHR(13));
AED3  140   WRITELN('CITUL=',C,'      ','RESTUL=',A);
AF1A  150   WRITELN(CHR(13))
AF21  160 END {$P}.
```

```
A=3        B=12
CITUL=0         RESTUL=3

A=27       B=12
CITUL=2         RESTUL=3

A=63       B=9
CITUL=7         RESTUL=0
```

P.III.9.

3.3.2. Instrucţiunea REPEAT

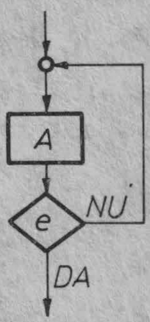


fig. 3. 11.

Forma generală a structurii repetitive de tipul DO-UNTIL este cea din fig.3.11.

Structura prezentată se descrie în PASCAL cu instrucţiunea REPEAT, a cărei sintaxă este dată de diagrama din fig.3.12.

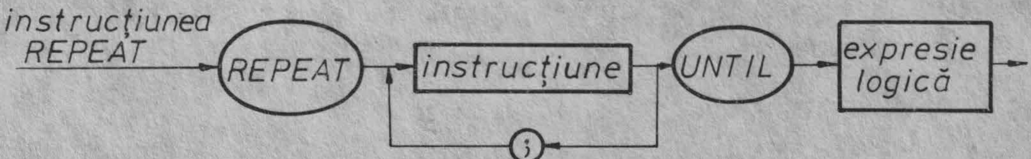


fig. 3. 12.

Efect:

-corpul ciclului (cuprins între REPEAT şi UNTIL) se execută *atît timp cît valoarea expresiei logice este falsă*; se iese din ciclu atunci cînd valoarea expresiei logice devine adevărat.

Observaţii:

1)- spre deosebire de instrucţiunea WHILE, în cazul instrucţiunii REPEAT corpul ciclului este executat *cel puţin o dată*;

2)-în anumite implementări, înaintea cuvîntului cheie UNTIL nu se pune ";"

3)- deoarece expresia logică se evaluează la fiecare iteraţie, ea trebuie să fie cît se poate de simplă.

Exemple:

a)
```
REPEAT

  X:=X+1
UNTIL X>0;
```

b)
```
REPEAT
  READ(A,B,C);
  F:=A*B*C;
  WRITE(F)
UNTIL F>100;
```

PROGRAMUL.III.1o.

Fie numărul real A, A>=0. Să se calculeze valoarea

rădăcinii pătrate a lui A cu o aproximaţie fixată prin constanta ε. Se va folosi formula de recurenţă:

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{A}{x_n}\right), \qquad n=0,1,2,\ldots \quad \text{unde } x_0 = A.$$

```
┌ algoritmul RAD este:
│  citeste a,ε
│  x1 ← a
│ ┌ repeta
│ │   x2 ← x1
│ │   x1 ← (x2+ a/x2)/2
│ └ iesimcînd (x2-x1)<ε
│  scrie x1
└ stop
```

Programul PASCAL corespunzător este **RAD** (P.III.10)

```
AE22   20 PROGRAM RAD;
AE22   30 CONST EPS=1E-8;
AE22   40 VAR A,X1,X2:REAL;
AE2B   50 BEGIN
AE34   60   READ(A);
AE3E   70   X1:=A;
AE4C   80   REPEAT
AE4C   90       X2:=X1;
AE5D  100       X1:=(X2+A/X2)/2
AE86  110   UNTIL (X2-X1)<EPS;
AEBC  120   WRITELN(CHR(13));
AEC6  130   WRITELN('RADACINA PATRATA DIN ',A:10:2,'=',X1:10:5);
AF16  140   WRITELN(CHR(13))
AF1D  150 END {$P}.

RADACINA PATRATA DIN      2.00=   1.41421

RADACINA PATRATA DIN     11.10=   3.33167

RADACINA PATRATA DIN    123.21=  11.10000

RADACINA PATRATA DIN      9.00=   3.00000
```

P.III.10

PROGRAMUL III.11.

Să se ordoneze crescător trei numere naturale.

```
┌ algoritmul ORD1 este:
│  citeste a,b,c
│ ┌ repeta
│ │   sw ← 0
│ │ ┌ daca a>b atunci sw ← 1
│ │ │                 x ← a
│ │ │                 a ← b
│ │ │                 b ← x
│ │ └ sfdaca
│ │ ┌ daca b>c atunci sw ← 1
│ │ │                 x ← b
│ │ │                 b ← c
│ │ │                 c ← x
│ │ └ sfdaca
│ └ iesimcînd sw=0
│  scrie a,b,c
└ stop
```

Programul de rezolvare a problemei este ilustrat în **ORD1** (P.III.11). Corpul ciclului este descris de liniile

90-180, care cuprind două structuri alternative simple. Valorile citite inițial sînt afișate pe prima linie, iar pe linia a doua sînt afișate aceleași valori ordonate crescător.

```
AD0F   20 PROGRAM ORD1;
AD0F   30 VAR A,B,C,SW,X:INTEGER;
AD18   40 BEGIN
AD21   50   READ(A,B,C);
AD33   60    WRITELN;
AD36   70    WRITELN('A=',A,'    B=',B,'    C=',C);
AD83   80    REPEAT
AD83   90       SW:=0;
AD8C  100       IF A>B THEN
AD9F  110          BEGIN
AD9F  120             SW:=1;X:=A;
ADAB  130             A:=B;B:=X
ADB1  140          END;
ADB7  150       IF B>C THEN
ADCA  160          BEGIN
ADCA  170             SW:=1;X:=B;
ADD6  180             B:=C;C:=X
ADDC  190          END
ADE2  200    UNTIL SW=0;
ADF4  210    WRITELN;WRITELN;
ADFA  220    WRITELN(A:9,B:9,C:9);
AE21  230 END {$P}.


A=42     B=947     C=29


        29       42       947
```

P.III.11.

PROGRAMUL III.12.

Să se afițeze histograma termenilor șirului Fibonacci, termeni care sînt mai mici sau egali decît o valoare maximă dată.

```
┌ algoritmul HIST este:
│   citeşte max
│   a ← 1
│   b ← 1
│  ┌ repeta
│  │   i ← 1
│  │  ┌ repeta
│  │  │   scrie '*'
│  │  │   i ← i+1
│  │  └ ieşimcînd i>a
│  │   c ← a+b
│  │   a ← b
│  │   b ← c
│  └ ieşimcînd a>max
└ stop
```

Exemplul descrie două structuri repetitive imbricate, ambele de tipul DO-UNTIL. Ciclul exterior descris de liniile 70-170 ale programului HIST (P.III.12). cuprinde la rîndul său un alt ciclu în liniile 90-120.

```
AE3C   10 PROGRAM HIST;
AE3C   20 VAR A,B,C,I,MAX:INTEGER;
AE45   30 BEGIN
AE4E   40   READ(MAX);
AE54   50   WRITELN(CHR(13));WRITELN;WRITELN('MAXIM:',MAX);
AE7E   60   A:=1;B:=1;
AE8A   70   REPEAT
AE8A   80      I:=1;
```

```
AE93      REPEAT
AE93        WRITE('*');
AE9B        I:=I+1
AEA3      UNTIL I>A;
AEB5      WRITELN;
AEB8      C:=A+B;
AEC9      A:=B;
AECF      B:=C
AECF    UNTIL A>MAX;
AEE8    WRITELN(CHR(13))
AEEF  END {$P}.


MAXIM:70
*
*
**
***
*****
********
*************
*********************
**********************************
*******************************************************
```

P.III.12.

3.3.3. Instrucţiunea FOR

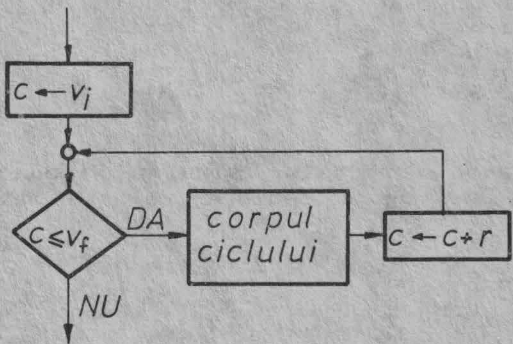


fig. 3. 13

Dacă numărul de repetiţii ale execuţiei structurii este cunoscut, atunci se recomandă structura repetitivă de tipul DO-FOR a cărei formă generală este cea din fig.3.13.

Variabila de control *c* (contorul) se iniţializează cu valoarea v_i şi creşte cu pasul *r* (raţia) pînă la valoarea v_f inclusiv.

In limbajul PASCAL, această structură se descrie cu ajutorul instrucţiunii FOR, a cărei sintaxă este dată în fig.3.14.

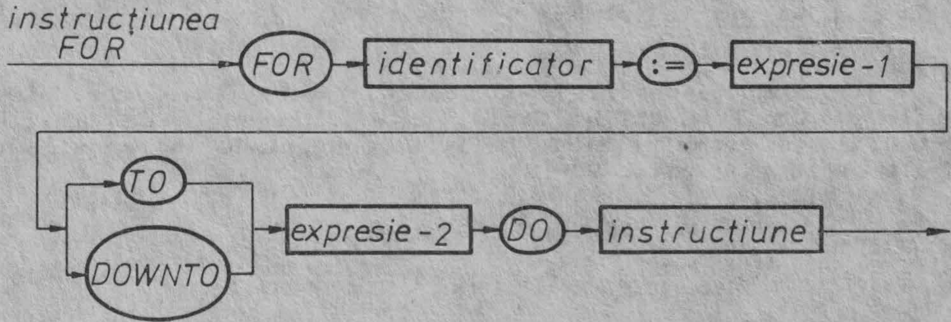


fig. 3. 14.

Corpul ciclului este instrucţiunea (evident, poate fi şi instrucţiune compusă sau un alt ciclu FOR) care urmează cuvîntului cheie DO, iar contorul ciclului este desemnat prin *identificator*.

Efect:

a) Forma cu **TO**

1)-se evaluează v_1, v_2 (valorile expresiilor *expresie-1* respectiv *expresie-2*);

2)-se iniţializează contorul *c* cu valoarea v_1;

3)-se compară valoarea contorului cu v_2:

-dacă $c>v_2$ se iese din instrucţiunea FOR;

-dacă $c<=v_2$ se execută *instrucţiune* şi se continuă cu 4;

4)-se calculează următoarea valoare a contorului, adică se incrementează *c* cu o unitate şi se reia pasul 3.

b) Forma cu **DOWNTO**

1)-se evaluează v_1, v_2 (valorile expresiilor *expresie-1* respectiv *expresie-2*);

2)-se iniţializează contorul *c* cu valoarea v_1;

3)-se compară valoarea contorului cu v_2:

-dacă $c<v_2$ se iese din instrucţiunea FOR;

-dacă $c>=v_2$ se execută *instrucţiune* şi se continuă cu 4;

4)-se calculează următoarea valoare a contorului, adică se decrementează *c* cu o unitate şi se reia pasul 3.

Observaţii:

1)-contorul şi expresiile din instrucţiunea FOR trebuie să fie de acelaşi tip *ordinal*;

2)-valorile ambelor expresii se determină *o singură dată* la începutul ciclării; ca urmare, modificarea în ciclu a valorilor variabilelor componente în aceste expresii *nu* afectează numărul de iteraţii;

3)-pentru majoritatea implementărilor, la ieşirea din ciclu valoarea contorului este *nedefinită*.

Pot exista următoarele situaţii (forma TO) :

a) $v_i>v_f$ - ciclul nu este executat niciodată;

b) $v_i=v_f$ - ciclul este executat numai o dată;

c) $v_i<v_f$ - ciclul este executat de (v_f-v_i+1) ori.

Evident, numărul execuţiilor se stabileşte în mod similar şi pentru forma cu DOWNTO.

Exemple:

```
a) FOR I:=1 TO N DO Y:=Y*I;

b) FOR I:=1 TO N DO
     BEGIN
       WRITE(' ');
       FOR J:=1 TO 2*I-1 DO WRITE('*')
     END;
```

Exemplul b) ilustrează două cicluri imbricate.

Programul.III.13.

Să se calculeze şi să se afişeze primii *i* termeni (*i*=1,10) ai sumei:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \ldots$$

```
algoritmul SIR este:
   s ← 1
   t ← 1
   pentru i=1,10 executa
      t ← t/i
      s ← s+t
      scrie s
   sfpentru
stop
```

Algoritmul este implementat în programul SIR (P.III.13) care afişează pe două coloane numărul de ordine al sumei şi valoarea corespunzătoare.

```
AC7D   10 PROGRAM SIR;
AC7D   20 VAR I:INTEGER;
AC86   30      S,T:REAL;
AC86   40 BEGIN
AC8F   50   S:=1;
AC9C   60   T:=1;
ACA9   70   FOR I:=1 TO 10 DO
ACC3   80        BEGIN
ACC6   90            T:=T/I;
ACDF  100            S:=S+T;
ACF9  110            WRITELN(I:2,S:10:5)
AD19  120        END;
AD1F  130 END {$P}.


 1   2.00000
 2   2.50000
 3   2.66667
 4   2.70833
 5   2.71667
 6   2.71805
 7   2.71825
 8   2.71828
 9   2.71828
10   2.71828
```

P.III.13.

PROGRAMUL III.14.

Se citesc pe rînd N numere reale. Să se calculeze şi să se afişeze media aritmetică a cîte două numere citite succesiv.

```
algoritmul MARIT este:
   citeste n,a
   pentru i=1,n-1 executa
      citeste b
      ma ← (a+b)/2
      scrie ma
      a ← b
   sfpentru
stop
```

Programul de rezolvare a problemei în PASCAL este MARIT (P.III.14).

```
AE66   20 PROGRAM MARIT;
AE66   30 VAR A,B,MA:REAL;
AE6F   40     I,N:INTEGER;
AE6F   50 BEGIN
AE78   60   READ(N,A);
AE88   70   WRITELN(CHR(16),CHR(13),'N=',N,CHR(13),CHR(16));
AEBD   80   FOR I:=1 TO N-1 DO
AEDC   90       BEGIN
AEDF  100           READ(B);
AEE9  110           MA:=(A+B)/2;
AF0E  120           WRITELN(CHR(16));
AF18  130           WRITELN('A=',A:6:2,'  ','B=',B:6:2);
AF6A  140           WRITELN('MA=',MA:6:2);
AF8F  150           WRITELN(CHR(16));
AF99  160           A:=B
AF99  170       END
AFA7  180 END {$P}.

N=7

A= 12.50  B= 34.00
MA= 23.25


A= 34.00  B= 34.00
MA= 34.00

A= 34.00  B=124.78
MA= 79.39

A=124.78  B= 67.30
MA= 96.04

A= 67.30  B=  7.95
MA= 37.62

A=  7.95  B= 20.00
MA= 13.97
```

P.III.14.

3.4. INSTRUCTIUNEA GOTO

Sintaxa instrucţiunii GOTO este dată de diagrama din fig.3.15.

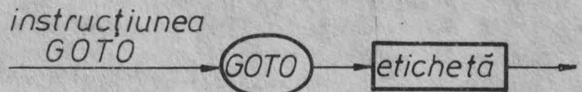


fig. 3. 15.

Efect:

-execuţia continuă *necondiţionat* cu instrucţiunea ce poartă eticheta din GOTO;

Etichetele se pot forma din maxim patru cifre zecimale şi se declară într-o declaraţie LABEL, înaintea declarării constantelor şi variabilelor.

Cînd o etichetă este folosită pentru a marca o instrucţiune, ea trebuie pusă la începutul instrucţiunii şi va fi urmată de ':'. Diagrama de sintaxă a declaraţiei de etichete este cea din fig.3.16.

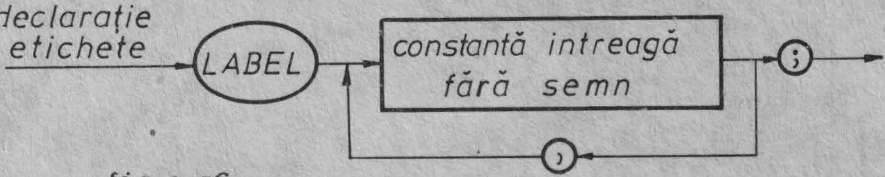


fig. 3. 16.

Exemple:

a)
```
LABEL 13;
........
BEGIN
.............
  13:A:=I+1;
.............
  GOTO 13;
.............
END
```

b)
```
LABEL 25;
.........
BEGIN
.........
  REPEAT
  ...........
    GOTO 25;
  ............
  UNTIL I>N;
25:WRITE(I)
............
END
```

Transferul controlului execuţiei este permis doar din cadrul unei instrucţiuni structurate *în afara* instrucţiunii (ex.b), *invers nu*. In acest sens, următorul exemplu este incorect:

```
LABEL 42;
............
BEGIN
............
  GOTO 42;
  ........
    BEGIN
    ........
     42:IF ...
     .........
    END;
    .........
END
```

Observaţie:

In general se recomandă evitarea utilizării instrucţiunii GOTO. Prezenţa ei se justifică doar în acele cazuri cînd din interiorul a mai multor structuri repetitive vrem să "evadăm".

PROGRAMUL.P.III.15.

Să se citească un număr oarecare de numere întregi. Să se calculeze media aritmetică a elementelor pe grupuri. Grupurile sînt precedate de un număr care reprezintă numărul elementelor din grup. Dacă acest număr este 0 sau negativ, nu avem grup. Elementele grupurilor trebuie să fie numere pozitive. Dacă un element este negativ sau 0, se semnalează eroare şi se opreşte execuţia.

```
┌algoritmul SUME este:
│  citeste n
│ ┌cittimp n>0 executa
│ │  suma ←— 0
│ │ ┌pentru i=1,n executa
│ │ │  citeste elem
│ │ │ ┌daca elem<=0
│ │ │ │  atunci scrie 'dată eronată'
│ │ │ │         stop
│ │ │ └sfdaca
│ │ │  suma ←— suma + elem
│ │ └sfpentru
│ │  medie ←— [suma/n]
│ │  scrie medie
│ │  citeste n
│ └sfcittimp
└stop
```

Cum un program PASCAL nu poate avea două instrucţiuni "END.", (corespunzătoare <u>stop</u>-ului din pseudocod) în caz de dată eronată trebuie să indicăm salt la sfîrşitul programului. Caracterul ';' din linia 200 a programului **SUME** (P.III.15) generează o instrucţiune vidă absolut necesară. Dacă ar lipsi, eticheta 1000 nu ar indica o instrucţiune.

```
AE1A   10 PROGRAM SUME;
AE1A   20 LABEL 1000;{ELEMENT ERONAT IN SUBSIR}
AE20   30 VAR N,I,ELEM,SUMA,MEDIE:INTEGER;
AE29   40 BEGIN
AE32   50   READ(N);{NUMAR ELEMENTE IN SUBSIR}
AE38   60   WHILE N>0 DO
AE4E   70     BEGIN
AE4E   80       SUMA:=0;
AE54   90       FOR I:=1 TO N DO
AE72  100         BEGIN
AE75  110           READ(ELEM);
AE7B  120           IF ELEM<=0 THEN
AE90  130             BEGIN
AE90  140               WRITELN('DATA ERONATA');
AEAA  150               GOTO 1000;
AEAD  160             END;
AEAD  170           SUMA:=SUMA+ELEM
AEB1  180         END;
AEC2  190       MEDIE:=SUMA DIV N;{REZULTAT ROTUNJIT}
AED1  200       WRITELN('MEDIA SUBSIRULUI CU ',N,' ELEMENTE POZITIVE ESTE '
AF18  210       ,MEDIE);
AF28  220       READ(N)
AF2E  230     END;
AF31  240   1000:{VALOARE ERONATA-NU DORIM SA O PRELUCRAM;
AF31  250         NU PUTEM AFISA NIMIC DEOARECE EVENTUALUL
AF31  260         MESAJ AR APAREA SI IN CAZUL DATELOR CORECTE}
AF31  270 END{$P}.
```

P.III.15.

Cum orice program poate fi scris fără GOTO, în continuare prezentăm varianta respectivă pentru programul de mai sus. Variabila booleeană *eroare* din programul **SUMEFARAGOTO** (P.III.16) permite evitarea prelucrării unei date eronate.

```
ADA5  PROGRAM SUMEFARAGOTO;
ADA5  VAR N,I,ELEM,SUMA,MEDIE:INTEGER;
ADAE      EROARE:BOOLEAN;
ADAE  BEGIN
ADB7    EROARE:=FALSE;
ADBB    READ(N);
ADC1    WHILE(N>0)AND NOT EROARE DO
ADDF      BEGIN
ADDF        SUMA:=0;
ADE5        I:=1;
ADEB        REPEAT
ADEB          READ(ELEM);
ADF4          IF ELEM>0
ADFC            THEN
AE07              BEGIN
AE07                SUMA:=SUMA+ELEM;
AE18                I:=I+1
AE20              END
AE1F            ELSE
AE22              EROARE:=TRUE;
AE27          UNTIL (I>N) OR EROARE;
AE40          IF EROARE
AE40            THEN
AE47              WRITE('EROARE LA INTRARE')
AE63            ELSE
AE66              BEGIN
AE66                WRITE('MEDIA ROTUNJITA A SUBSIRULUI DIN ');
AE92                WRITELN(N,' ELEMENTE ESTE ',SUMA DIV N);
AECA              END;
AECA        END
AECA  END{$P}.
```

P.III.16.

Comparînd cele două variante de rezolvare ale aceleiaşi probleme se observă că programul cu GOTO este mai lizibil şi mai scurt, deci *în acest caz* se justifică utilizarea acestei instrucţiuni.

PROBLEME PROPUSE

1. Fiind dată valoarea în radiani a unui unghi, să se calculeze valoarea echivalentă în grade, minute şi secunde.

2. Să se calculeze valoarea funcţiei f:R ⟶ R:

$$f(x,y) = \begin{cases} x+y & \text{dacă } 0<=x-y<=9 \\ x+y & \text{dacă } x-y<0,\ y>=0 \\ x-y & \text{în restul cazurilor} \end{cases}$$

pentru x, y ∈ R.

3. Se dau trei puncte prin coordonatele lor. Să se calculeze perimetrul şi aria triunghiului determinat de punctele date.

4. Să se calculeze produsul a două numere naturale prin adunări repetate.

5. Să se afişeze termenii şirului Fibonacci mai mici decît o valoare dată.

6. Să se calculeze cmmdc şi cmmmc a două numere nat... date, folosind algoritmul lui Euclid.

7. Fiind dat n natural, să se calculeze sumele :
a) s = 1 + 1*2 + 1*2*3 + ...+ 1*2*3* ...*n

b) $s = 1 + \frac{1}{1*2} + \frac{2}{1*2*3} + \ldots + \frac{n-1}{1*2*3*\ldots n}$

8. Se citesc pe rînd n numere reale. Să se calculeze produsul celor nenule şi suma celor mai mici decît 10.

9. Se citesc pe rînd n numere reale. Să se numere cîte dintre ele sînt pare şi pozitive.

IV. TIPURI DE DATE

Tipul unei date definește o *mulțime de valori* pe care le poate lua orice variabilă de tipul respectiv și *operațiile* care se pot efectua utilizînd aceste variabile.

Introducerea unui tip de date într-un program se face folosind *specificarea de tip*. Tipul nou introdus *poate purta un nume* sau poate rămîne *anonim*. In primul caz se utilizează definiția de tipuri din diagrama din fig.4.1.

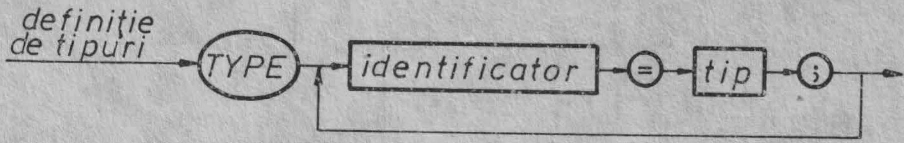


fig. 4. 1.

unde:

-*identificator* este numele asociat prin definiție tipului; identificatorul care a fost prezent într-o definiție de tip va desemna mulțimea valorilor tipului specificat și poate fi folosit ca și identificator de tip în alte definiții de tip sau în declararea variabilelor etc.;

-*tip* este explicitat în diagrama din fig.4.2, respectiv fig.4.3 și 4.4, tipurile de date urmînd a fi tratate în acest capitol și în capitolele următoare.

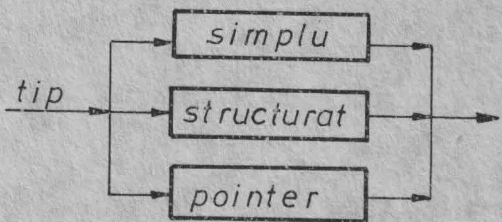


fig. 4. 2.

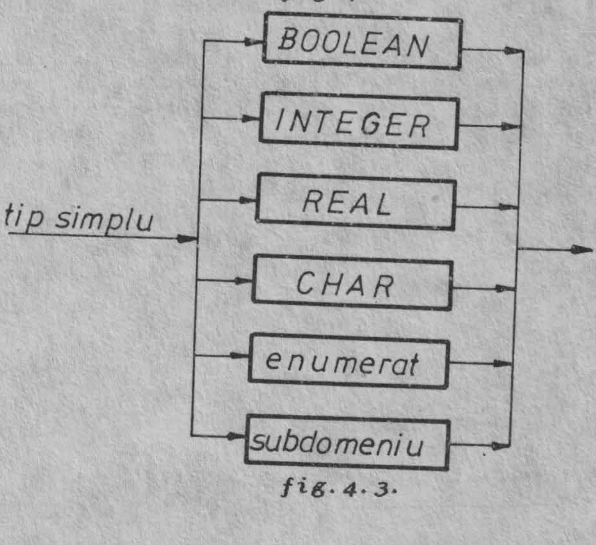


fig. 4. 3.

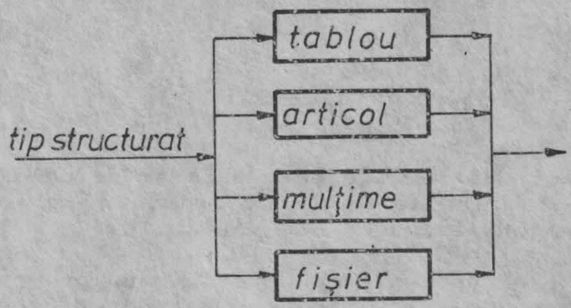


fig. 4. 4.

Identificatorul de tip poate fi referit în următoarele moduri:

a)-într-o altă definiţie de tip, obţinînd astfel structuri complexe;

b)-într-o declaraţie de variabile (capitolul II);

c)-într-o declaraţie de funcţie (capitolul V);

d)-într-o listă de parametri formali (capitolul VI).

Dacă dorim să introducem una sau mai multe variabile de un anumit tip, declararea acestuia se poate face direct în declaraţia de variabile, obţinînd în acest caz *tipuri anonime* de date. Declaraţia de variabile are forma dată în diagrama din fig.4.5.

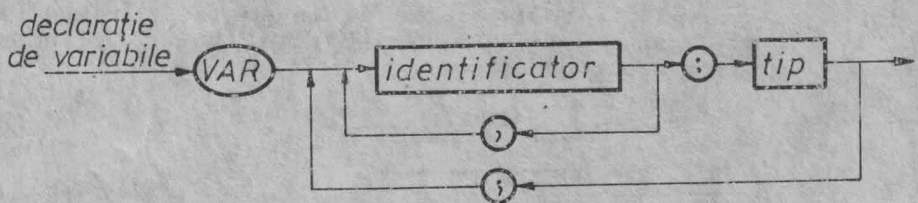


fig. 4. 5.

Observaţii:

1)-prin definirea tipului se realizează descrierea unei mulţimi posibile de valori, nu şi alocarea de zone de memorie, rezervarea realizîndu-se prin declararea unei variabile de tipul respectiv;

2)-definiţia de tipuri este situată în program după declaraţia etichetelor şi definiţia constantelor şi înaintea declaraţiei variabilelor şi a declaraţiilor subprogramelor;

3)-*In PASCAL două specificaţii de tip diferite introduc tipuri distincte.* (Asupra acestei observaţii se va reveni cu amănunte în acest capitol, pe măsură ce se tratează tipurile de date.)

4.1. TIPURI SIMPLE

In limbajul PASCAL, tipurile simple sînt de două feluri:

-*standard* (sau predefinite): întreg, real, logic, caracter;

-*definite de utilizator*: enumerat, subdomeniu.

Toate tipurile simple, cu excepţia celui real, se numesc tipuri *ordinale*.

Pentru a specifica un tip de date, în PASCAL se utilizează 6 modalităţi:

1) *Asimilarea cu anumite mulţimi de referinţă* (N,R). Acestor mulţimi infinite le corespunde în fiecare implementare a limbajului PASCAL cîte o submulţime finită de valori. Este cazul **tipurilor standard** INTEGER şi REAL. Pe lîngă acestea mai există şi tipurile predefinite CHAR reprezentînd o mulţime finită şi ordonată de caractere şi BOOLEAN care reprezintă mulţimea valorilor logice (TRUE şi FALSE).

2) *Enumerarea componentelor mulţimii*, obţinîndu-se **tipul enumerat**. In cadrul acestor mulţimi componentele sînt ordonate conform enumerării.

3) *Stabilirea unei submulţimi a unui tip ordinal*, în care fiecare element are un loc bine definit. Avînd în vedere că toate tipurile ordinale reprezintă o mulţime *cu cardinalitate cunoscută*, se poate lua în considerare o submulţime a acestora clar definită de elementul cel mai mic şi cel mai mare al submulţimii. Obţinem astfel un nou tip finit, numit **tip subdomeniu**. Tipul iniţial devine tipul asociat subdomeniului.

4) *Obţinerea unor tipuri structurate* pornind de la tipurile simple, *prin operaţiile*:

-*indexarea* cu un tip ordinal a unui tip de date deja definit (**tablou**);

-*compunerea* mai multor tipuri de date (**înregistrare**);

-*formarea mulţimii submulţimilor* de valori ale unui tip ordinal (**mulţime**);

-*înşiruirea* unui număr neprecizat de componente de acelaşi tip (**fişier**);

5) *Utilizarea unui tip definit în crearea* **tipurilor pointer** (capitolul VI.);

6) *Echivalarea cu un tip deja definit*, obţinînd o nouă denumire a tipului respectiv.

4.1.1. Tipuri simple standard

Cele patru tipuri simple standard sînt desemnate prin identificatorii predefiniţi: INTEGER, REAL, BOOLEAN şi CHAR.

Tipul întreg. Tipul întreg, desemnat prin identificatorul INTEGER semnifică o submulţime a numerelor întregi, cuprinsă între -MAXINT şi MAXINT.

In cazul implementării HP4TM, MAXINT = 32767.

In exemplul următor, variabilele NR1 şi NR2 sînt declarate de tip întreg: VAR NR1,NR2:INTEGER;

Operaţiile aritmetice definite între valori întregi sînt date de tabelul T.4.1.

Tabel T.4.1.

Operator	Operaţie
*	înmulţire
/	împărţire (rezultatul este real)
DIV	împărţire întreagă
MOD	restul împărţirii întregi
+	adunare
-	scădere

Observaţie:

Rezultatele operaţiilor asupra valorilor întregi reproduc rezultatele din aritmetica numerelor întregi.

Asupra valorilor întregi se pot aplica *operatorii relaţionali* cuprinşi în tabelul T.4.2.

Tabel T.4.2.

Operator	Operaţie
=	egal
<>	diferit
<	mai mic
>	mai mare
<=	mai mic sau egal
>=	mai mare sau egal

In limbajul PASCAL sînt definite *funcţii standard* care produc rezultate de tip întreg. Aceste funcţii sînt date în tabelul T.4.3.

Tabel T.4.3.

Nume funcţie	Argument	Semnificaţia rezultatului
ABS(X)	întreg	valoarea absolută a lui X
SQR(X)	întreg	pătratul lui X
SUCC(X)	întreg	succesorul lui X
PRED(X)	întreg	predecesorul lui X
ORD(X)	întreg	numărul de ordine al lui X
TRUNC(X)	real sau întreg	cel mai mare întreg mai mic sau egal cu X pentru X pozitiv sau cel mai mic întreg mai mare sau egal cu X dacă X este negativ
ROUND(X)	real sau întreg	cel mai apropiat întreg de X
ENTIER(X)	real sau întreg	cel mai apropiat întreg mai mic sau egal cu X

Observaţii:

1)-funcţiile TRUNC, ROUND şi ENTIER se numesc *funcţii de transfer* deoarece realizează transferul de la un tip (cel real) la un alt tip (cel întreg);

2)-dacă în urma operaţiilor aritmetice sau a aplicării funcţiilor, rezultatul care este atribuit unei variabile de tip întreg este în afara intervalului [-MAXINT, MAXINT], se semnalează eroare în timpul execuţiei;

3)-funcţia ENTIER este specifică implementării HP4TM.

In programul **FUNCINT** (P IV.1) sînt cuprinse exemple de utilizare a funcţiilor din tabelul T.4.3.

```
B150   10 PROGRAM FUNCINT;
B150   20 BEGIN
B162   30  WRITELN(CHR(16));
B16C   40  WRITELN('ABS(25)=',ABS(25):2);
B191   50  WRITELN('ABS(-25)=',ABS(-25));
B1B7   60. WRITELN('SQR(25)=',SQR(25));
B1DB   70  WRITELN('SUCC(25)=',SUCC(25));
B1FC   80  WRITELN('PRED(25)=',PRED(25));
B21D   90  WRITELN('TRUNC(25.6)=',TRUNC(25.5));
B246  100  WRITELN('ROUND(25.6)=',ROUND(25.6));
B26F  110  WRITELN('ENTIER(25.6)=',ENTIER(25.6));
B299  120  WRITELN('TRUNC(-25.6)=',TRUNC(-25.6));
B2CB  130  WRITELN('ROUND(-25.6)=',ROUND(-25.6));
```

```
B2FD  140  WRITELN('ENTIER(-25.6)',ENTIER(-25.6));
B32F  150  WRITELN('ROUND(-6.5)=',ROUND(-6.5));
B360  160  WRITELN('ROUND(-6.51)=',ROUND(-6.61));
B392  170  WRITELN('TRUNC(1.9)=',TRUNC(1.9));
B3BA  180  WRITELN('ROUND(1.9)=',ROUND(1.9));
B3E2  190  WRITELN('ENTIER(1.9)=',ENTIER(1.9));
B40B  200  WRITELN('TRUNC(-1.9)=',TRUNC(-1.9));
B43C  210  WRITELN('ENTIER(-1.9)=',ENTIER(-1.9));
B46E  220  WRITELN('ROUND(-1.9)=',ROUND(-1.9));
B49F  230  WRITELN('TRUNC(-1.1)=',TRUNC(-1.1));
B4D0  240  WRITELN('ENTIER(-1.1)=',ENTIER(-1.1));
B502  250  WRITELN('ROUND(-1.1)=',ROUND(-1.1));
B533  260  WRITELN('TRUNC(1.1)=',TRUNC(1.1));
B55B  270  WRITELN('ENTIER(1.1)=',ENTIER(1.1));
B584  280  WRITELN('ROUND(1.1)=',ROUND(1.1));
B5AC  285 WRITELN(CHR(16));
B5B6  290 END{$P}.
```

```
ABS(25)=25
ABS(-25)=25
SQR(25)=625
SUCC(25)=26
PRED(25)=24
TRUNC(25.6)=25
ROUND(25.6)=26
ENTIER(25.6)=25
TRUNC(-25.6)=-25
ROUND(-25.6)=-26
ENTIER(-25.6)-26
ROUND(-6.5)=-6
ROUND(-6.51)=-7
TRUNC(1.9)=1
ROUND(1.9)=2
ENTIER(1.9)=1
TRUNC(-1.9)=-1
ENTIER(-1.9)=-2
ROUND(-1.9)=-2
TRUNC(-1.1)=-1
ENTIER(-1.1)=-2
ROUND(-1.1)=-1
TRUNC(1.1)=1
ENTIER(1.1)=1
ROUND(1.1)=1
```

P.IV.1.

Programul **ALGEUCLID** (P.IV.2) determină cel mai mare divizor comun a două numere naturale utilizînd algoritmul lui Euclid. In acest program în linia 10 sînt declarate variabilele de tip întreg N, M, C, R. In linia 120 sînt utilizaţi operatorii -, DIV şi *.

```
AE36   10 PROGRAM ALGEUCLID;
AE36   20 VAR N,M,C,R:INTEGER;
AE3F   30 BEGIN
AE48   40   WRITE('N=');
AE55   50   READ(N);
AE5B   60   WRITE('M=');
AE68   70   READ(M);
AE6E   80   WRITELN;
AE71   90   WRITELN('N=',N);
AE8A  100   WRITELN('M=',M);
AEA3  110   REPEAT
AEA3  120     C:=N;
AEAC  130     R:=M MOD N;
AEBA  140     M:=N;
AEC0  150     N:=R;
AEC6  160   UNTIL R=0;
AED8  170   WRITELN('CMMDC=',C);
AEF5  180   WRITELN
AEF5  190 END {$P}.
```

```
N=6
M=120
CMMDC=6


N=7121
M=88
CMMDC=1


N=100
M=1230
CMMDC=10


N=88
M=65
CMMDC=1
```

P.IV.2.

In programul **IMPARTIRE** (P.IV.3) se calculează valoarea cîtului a două numere întregi cu un număr determinat de cifre în partea fracţionară. "Construirea" părţii fracţionare se realizează prin împărţiri succesive a restului obţinut.

```
AF1E  PROGRAM IMPARTIRE;
AF1E  { CONSTRUIREA CITULUI CU UN NUMAR DETERMINAT
AF1E    DE CIFRE LA PARTEA ZECIMALA }
AF1E  VAR A,B,C,N,I:INTEGER;
AF27  BEGIN
AF30   WRITE('A=');READ(A);
AF43   WRITE('B=');READ(B);
AF56   WRITE('N=');READ(N);
AF69   WHILE B<>0 DO
AF7E    BEGIN
AF7E     WRITELN(CHR(13));
AF88     WRITE('A=',A:3,'  B=',B:3,'  A/B=',A DIV B:3,'.':1);
AFF1     FOR I:=1 TO N DO
B00F      BEGIN
B012       A:=(A MOD B)*10;
B026       WRITE(A DIV B:1)
B03B      END;
B03F     WRITELN;
B042     WRITE(CHR(13));
B049     READLN;
B04C     WRITE('A=');READ(A);
B05F     WRITE('B=');READ(B);
B072     WRITE('N=');READ(N);
B085    END;
B088  END {$P}.
```

```
A=  5  B=  4  A/B=  1.250

A=123  B=  4  A/B= 30.7500000000

A= 32  B=  2  A/B= 16.0

A=  9  B= 65  A/B=  0.138461

A=  4  B=  5  A/B=  0.800000

A=  1  B=  3  A/B=  0.3333

A=  1  B= 10  A/B=  0.10000

A=633  B= 23  A/B= 27.52173913

A=532  B= 21  A/B= 25.333

A=  6  B=  5  A/B=  1.20
```

P.IV.3.

Tipul real. Tipul real este desemnat prin identificatorul predefinit REAL și este format dintr-o submulțime finită a numerelor raționale, cuprinsă între două limite ce diferă de la o implementare la alta.

Intervalul de valori în care se poate lucra utilizînd implementarea HP4TM este [-3.4E38, +3.4E38], însă numerele din vecinătatea lui 0, cuprinse între (-5.9E-39, 5.9E-39) se aproximează cu 0.

Datorită reprezentării în virgulă mobilă, *precizia este de 7 cifre semnificative*. De aceea nu are sens să utilizăm mai mult decît 7 cifre pentru specificarea mantisei unui număr real, întrucît cifrele în plus vor fi ignorate.

Exemplu:

Numărul 0.000123456 în reprezentarea internă pierde ultimele două cifre, deci este reprezentat mai puțin precis decît 1.23456E-4.

In timp ce rezultatele operațiilor asupra valorilor întregi sînt exacte, în cazul tipului real în cele mai multe cazuri, rezultatele sînt aproximative, din cauza rotunjirilor datorate folosirii unui număr finit de cifre la reprezentarea în virgulă mobilă.

Operațiile aritmetice definite între valori reale sînt date în tabelul T.4.4.

Tabel T.4.4.

Operator	Operație
*	înmulțire
/	împărțire
+	adunare
-	scădere

Operatorii relaționali menționați la tipul întreg sînt valabili și pentru tipul real. Rezultatul aplicării lor este tot o valoare logică.

Exemplu:

1.5>1.2 are valoarea TRUE;
1.5>2.0 are valoarea FALSE;
1.567896307=1.567896399 are valoarea TRUE.

Funcțiile standard care produc rezultate de tip real sînt date în tabelul T.4.5.

Programul **FUNCINT** (P.IV.1) conține cîteva exemple de utilizare a funcțiilor.

Observații:

1)-în cazul argumentelor întregi, la aplicarea funcțiilor SIN, COS, TAN, ARCTAN, LN, EXP, SQRT, se efectuează în prealabil *conversia* valorii argumentelor din întreg în real;

2)-operatorii aritmetici și unele funcții standard (EXP, SQR) pot provoca *depășire* flotantă;

3)-funcția FRAC este specifică implementării HP4TM;

4)-valorile reale *nu* pot fi folosite în următoarele cazuri:

- -argument pentru funcţiile standard SUCC, PRED şi ORD
- -index pentru tablouri;
- -contor pentru instrucţiunea FOR;
- -tip de bază pentru o mulţime;
- -selector în CASE;
- -definire tip subdomeniu.

Tabel T.4.5.

Nume	Tip argument	Semnificaţia rezultatului funcţiei
ABS(X)	real	valoarea absolută a argumentului
SQR(X)	real	pătratul valorii argumentului
SIN(X)	real sau întreg	sinusul valorii argumentului exprimat în radiani
COS(X)	real sau întreg	cosinusul valorii argumentului exprimat în radiani
TAN(X)	real sau întreg	tangenta valorii argumentului exprimată în radiani
ARCTAN(X)	real sau întreg	unghiul, exprimat în radiani, a cărui tangentă este X
LN(X)	real sau întreg	logaritm natural din X
EXP(X)	real sau întreg	exponenţiala valorii argumentului
SQRT(X)	real sau întreg	radical din X (X>=0)
FRAC(X)	real	partea fracţionară a lui X FRAC(X)=X-ENTIER(X)

Tipul logic. Tipul logic, desemnat prin identificatorul predefinit BOOLEAN este format din mulţimea valorilor logice TRUE şi FALSE, doi identificatori predefiniţi avînd valoarea logică *fals* respectiv *adevărat*.

Operatorii logici care se aplică asupra valorilor logice sînt:

a) operatorii binari:
   OR (disjuncţie logică);
   AND (conjuncţie logică);

b) operator unar:
   NOT (negaţie logică).

In mulţimea formată din elementele TRUE şi FALSE se introduce relaţia de ordine FALSE<TRUE. Asupra tipului logic se pot aplica operatorii relaţionali daţi în tabelul T.4.6.

Tabel T.4.6.

Operator	Semnificaţie
=	echivalenţă
<>	nonechivalenţă (*sau exclusiv*)
<=	implicaţie

Funcţiile standard care admit argumente de tip logic sînt date în tabelul T.4.7.

Tabel T.4.7.

Funcţia	Tipul valorii	Semnificaţia
ORD(X)	logic	numărul de ordine al argumentului X
PRED(X)	logic	predecesorul argumentului X
SUCC(X)	logic	succesorul argumentului X

Funcţiile standard care produc rezultate de tip logic sînt cuprinse în tabelul T.4.8.

Tabel T.4.8.

Funcţia	Tipul argumentului	Semnificaţia
ODD(X)	întreg	este TRUE pentru x impar
EOLN	-	este TRUE dacă valoarea care urmează valorii citite este caracterul *sfîrşit de linie*

Observaţie:

In implementările limbajului PASCAL în care există tipul fişier, funcţia EOLN se poate apela şi cu argument *fişier de text*; de asemenea, există funcţia EOF de argument *fişier* care va furniza valoarea TRUE dacă următorul caracter de la intrare este marca de sfîrşit de fişier.

Tipul caracter. Tipul caracter este desemnat prin identificatorul standard CHAR şi este format, în cazul implementării HP4TM, din *setul extins de caractere ASCII*.

Mulţimea caracterelor şi ordinea acestora în mulţime diferă de la un tip de calculator la altul datorită diferenţelor din hardware-ul sistemului de calcul. In cazul studiat, ordinea elementelor în mulţime este dată de codurile ASCII ale caracterelor.

Asupra valorilor de tip caracter se pot aplica operatorii relaţionali.

Funcţiile standard care produc ca rezultat un caracter sînt date în tabelul T.4.9.

Tabel T.4.9.

Funcţia	Argumentul	Semnificaţia rezultatului
CHR(X)	întreg 32<=X<=164	caracterul care are codul ASCII X
SUCC(X)	caracter	succesorul caracterului X în mulţimea caracterelor
PRED(X)	caracter	predecesorul caracterului X în mulţimea caracterelor
INCH	-	execută baleierea tastaturii şi dacă a fost apăsată o tastă, returnează caracterul reprezentat de această tastă, iar dacă nu, se returnează CHR(0)

Acestor funcţii li se alătură funcţia ORD, de argument caracter, care returnează codul ASCII al caracterului căruia i se aplică.

Se observă că au loc relaţiile:

ORD(CHR(I))=I   I - întreg din [32,164]
CHR(ORD(C))=C   C - caracter ASCII

Tabel T.4.10.

Cod ASCII	caracter	Cod ASCII	caracter	Cod ASCII	caracter
32	spaţiu	77	M	122	z
33	!	78	N	123	{
34	"	79	O	124	¦
35	#	80	P	125	}
36	$	81	Q	126	~
37	%	82	R	127	
38	&	83	S	128	spaţiu
39	'	84	T	129	
40	(	85	U	130	c
41	)	86	V	131	a
42	*	87	W	132	r
43	+	88	X	133	a
44	,	89	Y	134	c
45	-	90	Z	135	t
46	.	91	[	136	e
47	/	92	\	137	r
48	0	93	]	138	e
49	1	94	^	139	
50	2	95	_	140	g
51	3	96	`	141	r
52	4	97	a	142	a
53	5	98	b	143	f
54	6	99	c	144	i
55	7	100	d	145	c
56	8	101	e	146	e
57	9	102	f	147	
58	:	103	g	148	d
59	;	104	h	149	e
60	<	105	i	150	f
61	=	106	j	151	i
62	>	107	k	152	n
63	?	108	l	153	i
64	@	109	m	154	t
65	A	110	n	155	e
66	B	111	o	156	
67	C	112	p	157	
68	D	113	q	158	
69	E	114	r	159	
70	F	115	s	160	
71	G	116	t	161	
72	H	117	u	162	
73	I	118	v	163	
74	J	119	w	164	
75	K	120	x		
76	L	121	y		

In afară de codurile menţionate în tabelul T.4.10., există următoarele posibilităţi de utilizare a funcţiei CHR pentru generarea *codurilor de control*:

- -CHR(0) - semnifică şir de caractere vid;
- -CHR(8) - mută cursorul înapoi cu un pas pe ecran cu ştergerea caracterului;
- -CHR(12) - şterge ecranul sau execută salt la pagină nouă dacă ieşirea este comutată la imprimantă
- -CHR(13) - execută retur de car şi avans de rînd;
- -CHR(16) - comută ieşirea între ecran şi imprimantă.

In programul **VOCALE** (P.IV.4) avem exemple de utilizare a variabilelor de tip caracter. Acest program numără vocalele aflate într-un şir de caractere introdus de la tastatură.

```
AE21   10 PROGRAM VOCALE;
AE21   20 CONST P=CHR(16);
AE21   30 VAR C:CHAR;
AE2A   40     NA,NE,NI,NO,NU:INTEGER;
AE2A   50 BEGIN
AE33   60  NA:=0;NE:=0;NI:=0;NO:=0;NU:=0;
AE51   70  WRITELN(P,'INTRODU O LINIE CU LITERE',CHR(13),P);
AE89   80  READLN;
AE8C   90  WHILE NOT EOLN DO
AE98  100   BEGIN
AE98  110    READ(C);WRITE(P,C,P);
AEAE  120    CASE C OF
AEB1  130     'A':NA:=NA+1;
AEC0  140     'E':NE:=NE+1;
AECF  150     'I':NI:=NI+1;
AEDE  160     'O':NO:=NO+1;
AEED  170     'U':NU:=NU+1;
AEFC  180     'B','Z','.',',',' ',';':BEGIN END
AF1A  190  END;
AF1A  200  END;
AF1D  210  WRITELN(P);WRITELN;
AF28  220  WRITELN('NUMARUL DE VOCALE DIN LINIE:');
AF52  230  WRITELN(' A=',NA:2);
AF6F  240  WRITELN(' E=',NE:2);
AF8C  250  WRITELN(' I=',NI:2);
AFA9  260  WRITELN(' O=',NO:2);
AFC6  270  WRITELN(' U=',NU:2);
AFE3  280  WRITELN(P);
AFEB  290 END {$P}.

INTRODU O LINIE CU LITERE
ACEASTA ESTE LINIA CU LITERE

NUMARUL DE VOCALE DIN LINIE:
 A= 4
 E= 5
 I= 3
 O= 0
 U= 1
```

P.IV.4.

4.1.2. Tipuri simple definite de utilizator

Specificarea unui tip de date se poate realiza şi în următoarele moduri:

a)-enumerarea componentelor mulţimii, obţinînd astfel tipul *enumerat*;

b)-stabilirea unei submulţimi a unui tip simplu, diferit de cel real, obţinînd astfel tipul *subdomeniu*; tipul simplu iniţial se numeşte tipul asociat subdomeniului.

Tipul enumerat. Deseori, în programe se utilizează date numerice în locul unor valori nenumerice deoarece nu există posibilitatea de a le introduce astfel în program.

De exemplu, pentru zilele săptămînii putem stabili următoarea corespondenţă:

```
luni       - 1
marţi      - 2
miercuri   - 3
joi        - 4
vineri     - 5
sîmbătă    - 6
duminică   - 7
```

In acest context 1 reprezintă ziua de luni, 2 reprezintă ziua de marţi etc...

Pentru a evita astfel de codificări, în PASCAL s-a introdus un nou tip de date, tipul enumerat, sporind astfel gradul de înţelegere şi lizibilitatea programului.

Introducerea unui tip enumerat se realizează conform diagramei din fig.4.6.

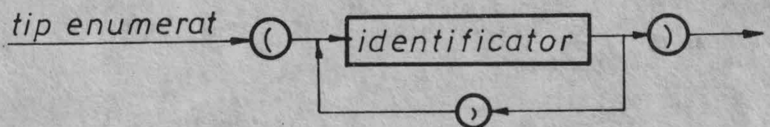


fig.4.6.

Identificatorii prezenţi între paranteze precizează valorile noului tip, putînd fi utilizate ca şi *constante* în program. Noul tip poate rămîne anonim sau poate primi un nume cînd se utilizează definiţia de tipuri. Numărul maxim de identificatori este 256.

Specificarea tipului enumerat introduce o mulţime ordonată de valori, ordine dată de locul identificatorului în cadrul listei. Identificatorii prezenţi într-o listă a unui tip enumerat nu pot fi utilizaţi în definirea unui alt tip enumerat, deoarece s-ar crea ambiguităţi cu privire la numărul de ordine al identificatorului respectiv.

Deoarece mulţimea valorilor unui tip enumerat este ordonată, între elementele sale se pot aplica operatorii relaţionali =, <, <>, <=, >=, >, rezultatul aplicării unui asemenea operator fiind o valoare logică.

Exemplu:

```
TYPE ZILE=(LUNI,MARTI,MIERCURI,JOI,VINERI,SIMBATA,DUMINICA);
VAR ZI:ZILE;
```

In exemplul anterior am definit un tip enumerat cu numele ZILE cu 7 elemente.

Considerînd definiţia de tipuri precedentă şi declaraţia de variabile, sînt permise atribuiri de forma:

```
ZI:=LUNI;
ZI:=DUMINICA;
```

Pentru tipul enumerat, ca şi pentru celelalte tipuri ordinale se pot aplica funcţiile standard din tabelul T.4.11.

Tabel T.4.11.

Funcţia	Semnificaţia
SUCC(X)	succesorul argumentului
PRED(X)	predecesorul argumentului
ORD(X)	are valoare întreagă şi determină rangul argumentului în enumrare, ştiind că primul element are rangul 0, iar între două elemente consecutive rangul creşte cu 1

In reprezentarea internă mărimile de tip enumerat sînt date de întregul corespunzător *numărului de ordine*. Deci

reprezentările interne sînt nişte întregi, însă spre deosebire de mărimile de tip întreg, între ele *nu se pot aplica* operatorii aritmetici şi deci nu pot apare ca şi operanzi în expresii aritmetice. Sînt permise doar atribuiri de elemente componente ale unui tip enumerat variabilelor declarate de tipul respectiv.

Observaţii:

1)-tipul BOOLEAN poate fi considerat ca un tip enumerat predefinit avînd ca şi elemente FALSE şi TRUE;

2)-valorile tipurilor enumerat *nu pot fi nici citite şi nici scrise*.

In programul **TIPENUM** (P.IV.5) sînt cîteva exemple de utilizare a tipului enumerat. Deoarece nu putem tipări o valoare de tip enumerat în aceste programe se tipăresc numerele de ordine ale acestor valori.

```
AE59    5 { TIPUL ENUMERARE }
AE59   10 PROGRAM TIPENUM;
AE59   20 TYPE ZILE=(LUNI,MARTI,MIERCURI,JOI,VINERI,SIMBATA,DUMINECA);
AE59   30 VAR ZI:ZILE;
AE62   40 BEGIN
AE6B   50  WRITELN(CHR(13));
AE75   60  WRITELN('ORD(LUNI)=',ORD(LUNI));
AE97   70  WRITELN('ORD(MARTI)=',ORD(MARTI));
AEBB   80  WRITELN('ORD(SUCC(JOI))=',ORD(SUCC(JOI)));
AEE4   85  WRITE(CHR(13));
AEEB   90 END{$P}.


ORD(LUNI)=0
ORD(MARTI)=1
ORD(SUCC(JOI))=4
```

P.IV.5.

Tipul subdomeniu. Deoarece uneori cunoaştem limitele în care o variabilă de tip ordinal poate lua valori, în PASCAL s-a introdus posibilitatea definirii tipului variabilei ca un interval al tipului ordinal respectiv. Acest nou tip poartă numele de subdomeniu sau interval.

Specificarea noului tip este dată în diagrama din fig.4.7.

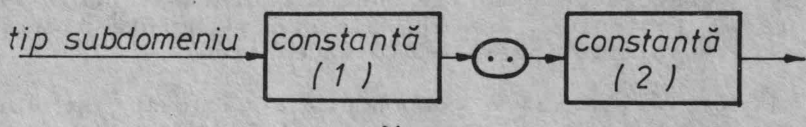


fig. 4.7.

unde:

-*constantă* reprezintă constantă sau identificator de constantă de tip ordinal, reprezentînd limita inferioară şi respectiv cea superioară a intervalului din care se alimentează variabilele de tipul subdomeniu.

Evident este necesar ca amîndouă constantele să fie de *acelaşi* tip ordinal şi prima constantă să fie strict mai mică decît a doua. Tipul celor două constante se numeşte *tip de bază* al subdomeniului.

Operatorii relaţionali şi aritmetici precum şi funcţiile standard corespunzătoare unui anumit tip ordinal sînt valabile pentru orice subdomeniu al său.

Se recomandă utilizarea acestui tip de date deoarece:

- pe parcursul rulării, în program se verifică încadrarea mărimii de tip subdomeniu între limitele declarate;

- utilizarea mărimilor de tip subdomeniu contribuie la creşterea clarităţii programului, prin explicitarea domeniului în care o variabilă ia valori;

- declararea componentelor de tip subdomeniu în cadrul unei structuri are ca efect reducerea spaţiului de memorie necesar reprezentării structurii.

Exemplu:

```
VAR A,B:INTEGER;
```

Dacă variabila A va avea în program valoarea minimă 1 şi cea maximă 100, iar B are valoarea minimă -10 şi cea maximă 1000, folosind tipul subdomeniu putem avea următoarea declaraţie:

```
VAR A:1..100;
    B:-10..1000;
```

4.2. TIPURI STRUCTURATE

In limbajul PASCAL pot fi definite structuri complexe pornind de la tipurile simple. Tipurile structurate, implementate în HP4TM sînt:

a)-*tipul tablou, rezultat din compunerea unor elemente de acelaşi tip, obţinînd astfel o structură omogenă;*

b)-*tipul înregistrare, rezultat din compunerea unor elemente de tipuri diferite, obţinînd astfel o structură neomogenă;*

c)-*tipul mulţime.*

4.2.1. Tipul tablou

Tabloul este un ansamblu omogen format dintr-un număr fix de elemente componente, toate fiind de acelaşi tip.

Tipul elementelor componente poartă numele de *tip de bază* al tabloului.

Ordinea elementelor în cadrul acestui tip este dată de asocierea acestora cu valorile unui tip ordinal, numit *tipul de indexare* al tabloului. Astfel, această structură transpune în programare mulţimile indexate din matematică.

Pentru a preciza o componentă a tabloului se utilizează numele acestuia şi o valoare a tipului de indexare numită *indice*, care indică locul elementului în cadrul ansamblului.

La definirea unui tablou trebuie precizate:

a)-tipul elementelor (tipul de bază al tabloului);

b)-tipul pentru indici (tipul de indexare).

Odată cu precizarea tipului pentru indici se fixează şi numărul componentelor tabloului care este dat de *cardinalitatea* tipului indicelui.

Intr-un program PASCAL, pentru a introduce un tablou vom folosi specificarea de tip conform diagramei din fig.4.8.

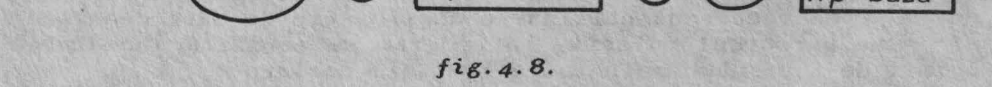


fig.4.8.

unde:

- *tip-indice* este specificarea tipului de indexare şi este un tip ordinal diferit de INTEGER: (enumerat, BOOLEAN, CHAR, subdomenii ale acestora, precum şi ale tipului INTEGER);
- *tip-bază* este tipul de bază al tabloului şi trebuie să fie cunoscut în momentul definiţiei acestuia.

O specificaţie de tip tablou reprezintă mulţimea tuturor tablourilor formate din elemente de tipul *tip-bază* indexate cu valori din mulţimea definită de *tip-indice*. Acest tip poate primi un nume (dacă este prezent în TYPE) sau poate rămîne anonim.

Cardinalitatea unui tip tablou este

$$n = a^b$$

unde: a = cardinalitatea tipului *tip-bază*;
b = cardinalitatea tipului *tip-indice*.

Pentru a putea lucra cu tablouri trebuie declarate variabile de tipul respectiv, operaţie ce trebuie realizată cu atenţie dacă dorim *manipularea tablourilor* în ansamblu. Aceasta constituie o facilitate a limbajului PASCAL faţă de multe alte limbaje de programare.

Reamintim faptul că *două specificaţii diferite de tip introduc tipuri distincte.*

Exemplu:

```
     TYPE SIR=ARRAY[1..10] OF INTEGER;
          SUBM=1..10;
     VAR  A,B:SIR;
          X,Y:ARRAY[SUBM] OF INTEGER;
```

In acest exemplu avem tipul structurat cu numele SIR care defineşte un tablou de 10 elemente numere întregi. Tipul de bază pentru tablou este INTEGER, iar tipul indicelui este subdomeniu de INTEGER. In declaraţia de variabile sînt declarate A şi B de tip SIR (adică şiruri de cîte 10 elemente numere întregi). In aceeaşi declaraţie de variabile sînt declarate X şi Y ca şiruri de 10 elemente numere întregi deoarece SUBM este un subdomeniu de întregi cu 10 elemente. Tipul variabilelor X şi Y este anonim. Deşi aparent A şi B au acelaşi tip cu X şi Y, ţinînd cont de observaţia că două specificaţii diferite de tip introduc tipuri diferite, rezultă că A şi B au un tip, iar X şi Y alt tip. In consecinţă este corectă o atribuire de felul:

```
A:=B; sau X:=Y;
```

dar nu este corectă o atribuire de genul:

```
A:=X; sau Y:=B;
```

O *componentă a tabloului se selectează* utilizînd numele variabilei tablou urmat de o expresie numită *expresie indiceală* încadrată de paranteze drepte, aşa cum precizează diagrama din fig.4.9.

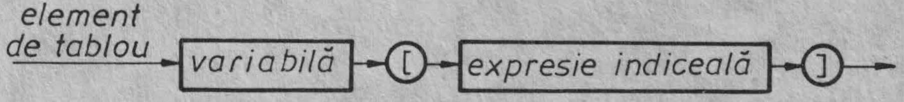


fig.4.9.

unde:

-*expresie indiceală* este o expresie a cărei valoare trebuie să aparţină tipului de indexare.

Deoarece pentru tip-bază nu s-a impus o restricţie, el poate fi, la rîndul lui, un tip tablou. Astfel, specificarea de tablou din exemplul următor se referă la un tablou cu elemente tablouri, adică la tablouri cu două dimensiuni. Mai mult, cînd tipul de bază este un tablou cu două dimensiuni obţinem un tablou cu trei dimensiuni etc...

Exemplu:

VAR A:ARRAY[1..10] OF ARRAY[-1..5] OF INTEGER;

introduce variabila A ca tablou cu 10 elemente, fiecare element fiind la rîndul său un tablou cu 7 elemente numere întregi, adică un tablou bidimensional.

Pentru a simplifica scrierea în cazul tablourilor cu mai multe dimensiuni, se recomandă ca specificaţia tipului tablou să se facă conform diagramei din fig.4.10.

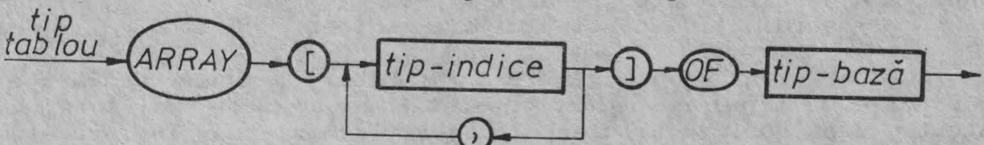


fig. 4. 10.

Referirea unui element se face cu ajutorul unei variabile indexate a cărei sintaxă este dată de diagrama din fig.4.11.

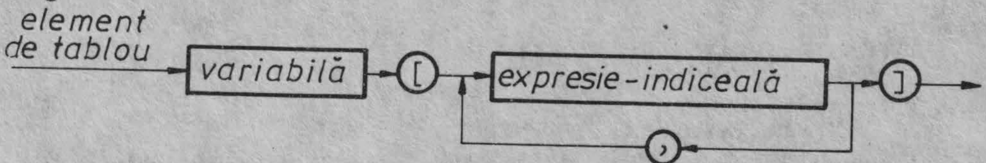


fig. 4. 11.

Pentru a referi un element, în HP4TM este permisă atît scrierea:

id-tablou[ind-1,ind-2,...,ind-n]

cît şi

id-tablou[ind-1][ind-2]...[ind-n]

Programul **TABLCHAR** (P.IV.6) roteşte cu 90° o figură generată din caractere.

```
AF91   PROGRAM TABLCHAR;
AF91   CONST N=8;
AF91   VAR I,J:INTEGER;
AF9A        M:ARRAY [1..64] OF CHAR;
AF9A        A,B:ARRAY [1..N,1..N] OF CHAR;
AF9A   BEGIN
AFA3   M:=' **  **   *  *   ****** ********** ** ** ******   *  *   **  **
AFF3   FOR I:=1 TO 8 DO
B00D    FOR J:=1 TO 8 DO
B02A    A[I,J]:=M[(I-1)*8+J];
B08F   (  FOR I:=0 TO N DO
B08F      FOR J:=0 TO N DO READ(A[I,J]))
B08F
B08F   {TIPARIRE A[I,J]}
B08F
B08F    PAGE;
B094     FOR I:=1 TO N DO
B0AE     BEGIN
B0B1      FOR J:=1 TO N DO WRITE(A[I,J]);
B106      WRITELN;
B109     END;
B10C
B10C   {ROTIRE CU 90 GRADE}
B10C
```

```
B10C  250  FOR I:=1 TO N DO
B126  260   FOR J:=1 TO N DO B[I,J]:=A[J,I];
B1B2  270
B1B2  280  WRITELN;WRITELN;
B1B8  290 (TIPARIRE B[I,J])
B1B8  300
B1B8  310   FOR I:=1 TO N DO
B1D2  320   BEGIN
B1D5  330    FOR J:=1 TO N DO WRITE(B[I,J]);
B22A  340    WRITELN;
B22D  350   END;
B230  360 END {$P}.
```

```
 **  **
  *  *
 ******
********
** ** **
 ******
  *  *
 **  **


   **
* **** *
**** ***
  ****
  ****
**** ***
* **** *
   **
```

P.IV.6.

Un concept important folosit în PASCAL este *împachetarea* structurilor de date, introdusă pentru economisirea memoriei. Impachetarea presupune rezervarea de către compilator a unui spaţiu de memorie minim structurii de date respective. Avantajul împachetării apare în cazul calculatoarelor cu lungime mare a cuvîntului (16, 32 biţi). Avînd în vedere că implementarea HP4TM este concepută pentru calculatoare personale şi acestea funcţionează cu microprocesor pe 8 biţi, împachetarea este inefectivă.

Din punct de vedere sintactic, o structură împachetată se introduce punînd cuvîntul rezervat PACKED înaintea declaraţiei.

Exemplu:

```
TYPE A=PACKED ARRAY[1..100] OF CHAR;
VAR Y:A;
    X:PACKED ARRAY[1..10] OF CHAR;
```

In acest exemplu sînt declarate două variabile tablou: Y de 100, respectiv X de 10 elemente caractere, amîndouă împachetate.

In programul **FIBONACCI** (P.IV.7) se generează şirul lui FIBONACCI.

```
AD76   10
AD76   20 { RULAREA PE IMPRIMANTA }
AD76   30
AD76   40 PROGRAM FIBONACCI;
AD76   50 VAR N:INTEGER;
AD7F   60     A:ARRAY[1..100] OF INTEGER;
AD7F   70 BEGIN
AD88   80   WRITELN(CHR(13));
AD92   90   WRITELN('GENERAREA PRIMILOR 20 DE TERMENI DIN SIRUL ');
ADCB  100   WRITELN('               F I B O N A C C I ');
ADF5  110   A[1]:=1;A[2]:=2;
AE3B  120   WRITELN('A( 1)=',A[1]:5);WRITELN('A( 2)=',A[2]:5);
```

```
AEB1      FOR N:=3 TO 20 DO
AECB       BEGIN
AECE        A[N]:=A[N-2]+A[N-1];
AF35        WRITELN('A(',N:2,')=',A[N]:5);
AF85       END;
AF88  END {$P}.
```

```
GENERAREA PRIMILOR 20 DE TERMENI DIN SIRUL
          F I B O N A C C I
A( 1)=    1
A( 2)=    2
A( 3)=    3
A( 4)=    5
A( 5)=    8
A( 6)=   13
A( 7)=   21
A( 8)=   34
A( 9)=   55
A(10)=   89
A(11)=  144
A(12)=  233
A(13)=  377
A(14)=  610
A(15)=  987
A(16)= 1597
A(17)= 2584
A(18)= 4181
A(19)= 6765
A(20)=10946
```

P.IV.7.

Programul **ADUNSIR** (P.IV.8) realizează suma a două şiruri.

```
AD53   PROGRAM ADUNSIR;
AD53   TYPE SIR=ARRAY[1..10] OF INTEGER;
AD53   VAR SIR1,SIR2,SUMA:SIR;
AD5C       I:INTEGER;
AD5C   BEGIN
AD65    WRITELN('INTRODU SIRURILE');
AD83    FOR I:=1 TO 10 DO
AD9D     BEGIN
ADA0      READ(SIR1[I],SIR2[I]);
ADE6      SUMA[I]:=SIR1[I]+SIR2[I];
AE4A      WRITELN(CHR(16));
AE54      WRITE('SIR1(',I:2,')=',SIR1[I]:6,' SIR2(',I:2,')=');
AECE      WRITELN(SIR2[I]:6,'   S(',I:2,')=',SUMA[I]:6);
AF48      WRITELN(CHR(16));
AF52     END
AF52   END {$P}.
```

```
SIR1( 1)=     1 SIR2( 1)=     1   S( 1)=     2

SIR1( 2)=     0 SIR2( 2)=     0   S( 2)=     0

SIR1( 3)= -1000 SIR2( 3)=  1000   S( 3)=     0

SIR1( 4)=   500 SIR2( 4)=   500   S( 4)=  1000

SIR1( 5)=     0 SIR2( 5)=     0   S( 5)=     0

SIR1( 6)= 12345 SIR2( 6)= 12345   S( 6)= 24690

SIR1( 7)=-12345 SIR2( 7)=  1000   S( 7)=-11345

SIR1( 8)=     9 SIR2( 8)=     0   S( 8)=     9

SIR1( 9)=     9 SIR2( 9)=     9   S( 9)=    18

SIR1(10)=     9 SIR2(10)=    -9   S(10)=     0
```

P.IV.8.

Programul **CONTORIZARE** (P.IV.9) este un exemplu de utilizare a indicilor de tip caracter.

```
AE88   10 { EXEMPLU CU INDICI LITERE}
AE88   20 PROGRAM CONTORIZARE;
AE88   30 VAR A:ARRAY['A'..'I'] OF REAL;
AE91   40     POZ:INTEGER;
AE91   50     I:CHAR;
AE91   60 BEGIN
AE9A   70  WRITELN('INTRODU SIRUL');
AEB5   80  POZ:=0;
AEBB   90  FOR I:='A' TO 'I' DO
AECB  100   BEGIN
AECE  110    READ(A[I]);
AF02  120    WRITELN('A(',I,')=',A[I]:7:2);
AF60  130    IF A[I] > 0
AF98  140          THEN
AFAD  150           POZ:=POZ+1;
AFB4  160   END;
AFC0  170  WRITELN;
AFC3  180  WRITELN('NUMARUL ELEMENTELOR POZITIVE=',POZ);
AFF7  190 END {$P}.
```

```
INTRODU SIRUL
A(A)=   3.66
A(B)=   8.23
A(C)=   6.00
A(D)=  -7.67
A(E)=   6.66
A(F)=  -2.21
A(G)=   1.23
A(H)= -45.67
A(I)= 111.11

NUMARUL ELEMENTELOR POZITIVE=6
```

P.IV.9.

Programul **INMAT** (P.IV.10) calculează produsul a două matrici cu elemente întregi.

```
AFF2   10 PROGRAM INMMAT;
AFF2   20 VAR A:ARRAY [1..2,1..4] OF INTEGER;
AFFB   30     B:ARRAY [1..4,1..3] OF INTEGER;
AFFB   40     C:ARRAY [1..2,1..3] OF INTEGER;
AFFB   50     I,J,K:INTEGER;
AFFB   60
AFFB   70 BEGIN
B004   80  WRITE(CHR(13));
B00B   90  PAGE;WRITELN;
B013  100  WRITE('INTRODUCETI MATRICEA A');
B034  110  WRITELN;
B037  120  FOR I:=1 TO 2 DO
B051  130   FOR J:=1 TO 4 DO
B06E  140    BEGIN
B071  150     WRITE('A[',I:1,',',J:1,']=');READ(A[I,J]);
B0E4  160    END;
B0EA  170  WRITELN;
B0ED  180  WRITE('INTRODUCETI MATRICEA B');
B10E  190  WRITELN;
B111  200  FOR I:=1 TO 4 DO
B12B  210   FOR J:=1 TO 3 DO
B148  220    BEGIN
B14B  230     WRITE('B[',I:1,',',J:1,']=');READ(B[I,J]);
B1C0  240    END;
B1C6  250  FOR I:=1 TO 2 DO
B1E0  260    FOR K:=1 TO 3 DO
B1FD  270     BEGIN
B200  280      C[I,K]:=0;
B23E  290      FOR J:=1 TO 4 DO
B258  300      C[I,K]:=C[I,K]+A[I,J]*B[J,K];
B34F  310     END;
B355  320
B355  330  WRITELN;
B358  340  WRITE('MATRICEA C');
```

```
B36D  350  FOR I:=1 TO 2 DO
B387  360   BEGIN
B38A  370    WRITELN;
B38D  380    FOR J:=1 TO 3 DO WRITE(C[I,J]:4);
B3EF  390   END;
B3F2  400  WRITELN;
B3F5  410 END {$P}.

INTRODUCETI MATRICEA A
A[1,1]=5
A[1,2]=3
A[1,3]=4
A[1,4]=7
A[2,1]=8
A[2,2]=1
A[2,3]=2
A[2,4]=3

INTRODUCETI MATRICEA B
B[1,1]=7
B[1,2]=5
B[1,3]=4
B[2,1]=3
B[2,2]=2
B[2,3]=1
B[3,1]=1
B[3,2]=2
B[3,3]=3
B[4,1]=5
B[4,2]=8
B[4,3]=5

MATRICEA C

MATRICEA C
  83  95  70
  76  70  54
```

P.IV.10.

Programul **TRIPASCAL** (P.IV.11) afişează primele 13 nivele ale triunghiului Pascal.

```
AE24   10 PROGRAM TRIPASCAL;
AE24   20 CONST N=13;
AE24   30 VAR I,J:INTEGER;
AE2D   40     A:ARRAY [1..N] OF INTEGER;
AE2D   50 BEGIN
AE36   60  FOR I:=1 TO N DO
AE50   70    BEGIN
AE53   80    A[I]:=1;
AE76   90    FOR J:=I-1 DOWNTO 2 DO
AE90  100       A[J]:=A[J-1]+A[J];
AEFB  110    WRITE(' ':2*(N-I)+1);
AF1F  120    FOR J:=1 TO I DO
AF3D  130       WRITE(A[J]:4);
AF6B  140    WRITELN
AF6B  150    END
AF6E  160 END {$P}.
```

```
                          1
                        1   1
                      1   2   1
                    1   3   3   1
                  1   4   6   4   1
                1   5  10  10   5   1
              1   6  15  20  15   6   1
            1   7  21  35  35  21   7   1
          1   8  28  56  70  56  28   8   1
        1   9  36  84 126 126  84  36   9   1
      1  10  45 120 210 252 210 120  45  10   1
    1  11  55 165 330 462 462 330 165  55  11   1
  1  12  66 220 495 792 924 792 495 220  66  12   1
```

P.IV.11.

4.2.2. Tipul înregistrare

Tipul înregistrare desemnează o structură formată dintr-un număr fix de componente (nu neapărat de acelaşi tip), numite *cîmpuri*.

Pentru a avea acces la elementele componente ale unei structuri de tip înregistrare, se folosesc *selectori*, (denumiri ale componentelor tipului respectiv).

Declararea unui tip structurat ca înregistrare este dată de diagrama din fig. 4.12.

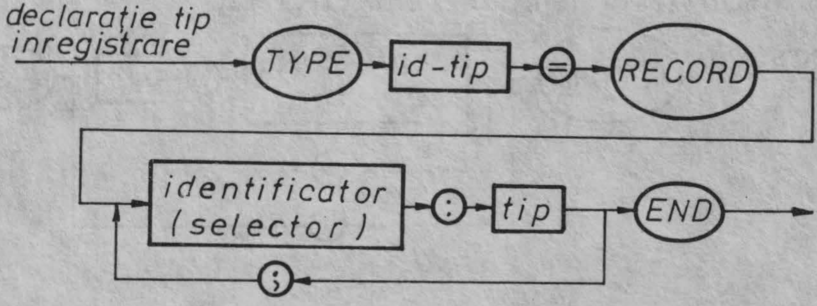


fig. 4.12.

caz în care noul tip primeşte şi un nume. Dacă nu se doreşte identificarea tipului printr-un nume, vom specifica tipul structurat ca înregistrare în declaraţia de variabile aşa cum rezultă din diagrama de sintaxă din fig. 4.13.

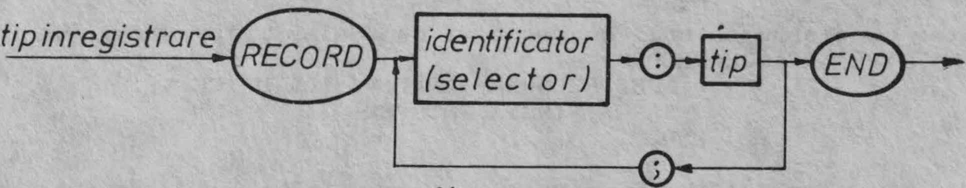


fig. 4.13.

caz în care noul tip rămîne anonim.

Observaţii:

1)-în cazul existenţei mai multor selectori, aceştia trebuie să fie *distincţi* pentru a identifica cîmpurile din cadrul structurii respective;

2)-*selectorii nu sînt calculabili şi nu li se pot atribui valori.*

Ca şi în cazul celorlalte tipuri, pentru a putea lucra cu o înregistrare trebuie declarată o variabilă de tipul respectiv.

Exemplu:

```
TYPE INREGISTRARE=RECORD
                    NUME:ARRAY[1..20] OF CHAR;
                    MATR:1..999
                  END;
```

Componentele unui tip înregistrare pot fi, la rîndul lor, structurate ca înregistrare.

Exemplu:

```
    TYPE ELEV=RECORD
                NUME:ARRAY[1..20] OF CHAR;
                DATA:RECORD
                       AN:1900..2000;
                       LU:1..12;
                       ZI:1..31
                     END;
                ADRESA:ARRAY[1..30] OF CHAR
              END;
```

Dacă două sau mai multe componente succesive sînt de acelaşi tip, tipul lor poate fi specificat o singură dată, precedat de lista selectorilor cîmpurilor identice, aşa cum rezultă din diagrama de sintaxă din fig.4.14.

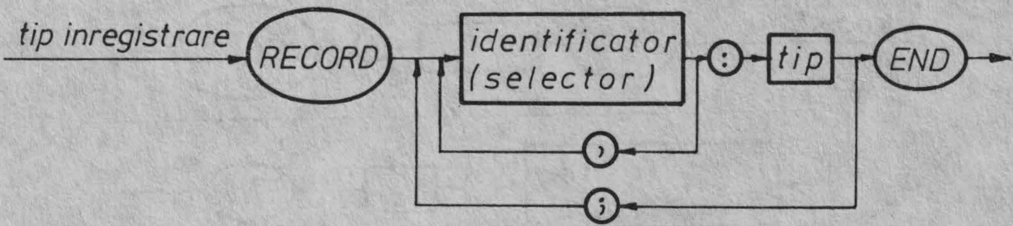


fig. 4. 14.

Exemplu:

Pentru următoarea structură de tip înregistrare

```
    TYPE ARTICOL=RECORD
                   NUME:ARRAY[1..10] OF CHAR;
                   PRENUME:ARRAY[1..10] OF CHAR;
                   MARCA:INTEGER;
                   VIRSTA:INTEGER
                 END;
```

putem cumula selectorii consecutivi de acelaşi tip astfel:

```
    TYPE ARTICOL=RECORD
                   NUME,PRENUME:ARRAY[1..10] OF CHAR;
                   MARCA,VIRSTA:INTEGER
                 END;
```

Accesul la o anumită componentă a unei variabile înregistrare se face prin intermediul *specificatorului de cîmp* care este o expresie formată din numele variabilei şi numele selectorului, după modelul din diagrama din fig.4.15.

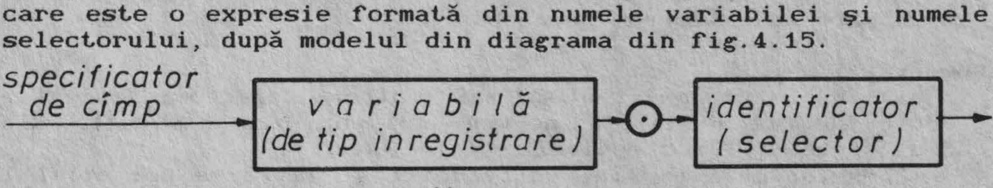


fig. 4. 15.

Pentru tipul înregistrare ELEV definit mai sus, cu declaraţia de variabile:

```
    VAR E:ELEV;
```

putem avea următoarele variabile selectate:

```
    E.NUME
    E.DATA.AN
    E.ADRESA
```

Observăm că în cazul în care o componentă a unui tip structurat ca înregistrare este la rîndul ei structurată ca înregistrare, selectorii şi expresiile indiceale se adaugă unul după celălalt pe măsură ce se pătrunde în structură.

In cazul utilizării înregistrărilor în care anumite cîmpuri sînt tot înregistrări, deci structurate pe mai multe nivele, accesul la o componentă încarcă foarte mult textul programului. Aceste cazuri, în PASCAL se rezolvă utilizînd instrucţiunea WITH. Sintaxa acestei instrucţiuni este dată de diagrama din fig.4.16.

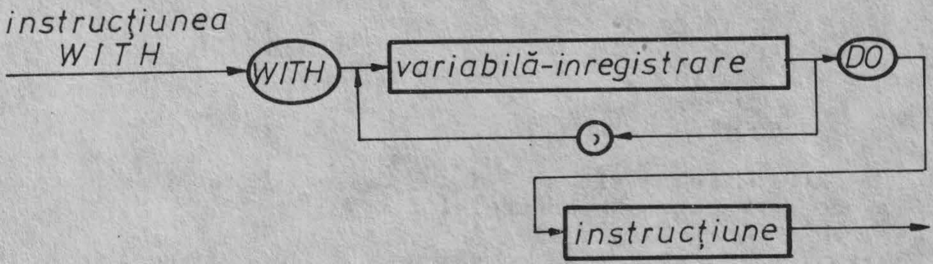


fig. 4. 16.

unde:

-*instrucţiune* este orice instrucţiune PASCAL.

Efectul instrucţiunii WITH se manifestă în faza de compilare, deci se referă la textul programului şi nu la execuţia lui. Instrucţiunea WITH delimitează în textul programului o zonă ce cuprinde textul lui *instrucţiune*. Peste această zonă variabilele din lista de variabile a instrucţiunii WITH acţionează astfel: în toate referinţele la componentele unei variabile *V* cuprinsă în listă, *se poate omite* prefixarea lor cu *V*, utilizînd chiar selectorii cîmpurilor respective. Aceştia pot apare, deci, în cadrul instrucţiunii sau blocului din cadrul lui WITH ca identificatori de variabile.

Exemplu:

```
VAR ELEV:RECORD
           NUME:ARRAY[1..20] OF CHAR;
           DATA:ARRAY[1..3] OF INTEGER;
           NOTA:REAL
         END;
```

Considerînd declaraţia de variabile anterioară, secvenţa:

```
READ(ELEV.NUME);
READ(ELEV.NOTA);
```

poate fi înlocuită, utilizînd instrucţiunea WITH, cu:

```
WITH ELEV DO
  BEGIN
    READ(NUME);
    READ(NOTA)
  END;
```

Observaţii:

1)-instrucţiunea

```
WITH A,B DO instrucţiune
```

este echivalentă cu grupul de instrucţiuni:

```
WITH A DO
  WITH B DO instrucţiune
```

2)-un selector dintr-o înregistrare poate avea acelaşi nume cu o variabilă oarecare din program;

Exemplu:

```
VAR A:CHAR;
    INREG:RECORD
            A:INTEGER;
            B:REAL
          END;
      .................
      WITH INREG DO
        BEGIN
          A:=10;
          B:=5.2
        END;
```

Atribuirea A:=10 se referă la specificatorul de cîmp INREG.A deoarece este sub controlul lui WITH;

3)-variabila de tip înregistrare nu poate fi schimbată în timpul execuţiei instrucţiunii WITH; în cazul în care avem un tablou cu elemente de tip înregistrare

```
VAR CLASA:ARRAY[1..40] OF RECORD
                            NUME:ARRAY[1..10] OF CHAR;
                            NUMAR:INTEGER
                          END;
```

următoarea secvenţă este nepermisă:

```
WITH CLASA[I] DO
  BEGIN
  .........
    I:=I+1;
  .........
  END;
```

deoarece mărirea variabilei I înseamnă schimbarea variabilei înregistrare.

In programul **RECOR** (P.IV.12) se declară o structură de tip înregistrare şi se exemplifică modul de lucru şi accesul la componentele variabilei. Tabloul ale cărui elemente sînt structurate ca înregistrare, reprezintă elevii unei clase. Sînt afişaţi acei elevi care au media peste 7. Informaţiile despre fiecare elev constau în nume, adresă, număr matricol şi medie.

```
B09F  PROGRAM RECOR;
B09F  TYPE DATE=RECORD
B09F       NUME,ADR:ARRAY [1..10] OF CHAR;
B09F       NR:INTEGER;
B09F       NOTA:REAL
B09F     END;
B09F  VAR ELEV:ARRAY[1..10] OF DATE;
B0A8      I,J:INTEGER;
B0A8
B0A8  BEGIN
B0B1   FOR I:=1 TO 4 DO
B0CB    BEGIN
B0CE     WRITELN('NUMELE:');READLN;READ(ELEV[I].NUME);
B10A     WRITELN('ADR:');READLN;READ(ELEV[I].ADR);
B147     FOR J:=1 TO 10 DO
B161      BEGIN
B164       IF ELEV[I].NUME[J]=CHR(0) THEN ELEV[I].NUME[J]:=' ';
B1E2       IF ELEV[I].ADR[J]=CHR(0) THEN ELEV[I].ADR[J]:=' ';
B268      END;
B26B     WRITE('NR:');READ(ELEV[I].NR);
B2A5     WRITE('NOTA:');READ(ELEV[I].NOTA);
B2E6     WRITELN;
B2E9    END;
B2EC
```

```
B2EC  FOR I:=1 TO 4 DO
B306   WITH ELEV[I] DO
B330    BEGIN
B330     WRITE(I,NUME,ADR:11,NR:3,NOTA:6:2);
B387     WRITELN
B387    END;
B38D   WRITELN;
B390   WRITELN('ELEVII CU NOTA MAI MARE DE 7');
B3BA  FOR I:=1 TO 4 DO
B3D4   IF ELEV[I].NOTA>=7.00 THEN
B41B    BEGIN
B41B     WRITELN('ELEVUL:',I:2);
B43C     WRITELN;
B43F     WITH ELEV[I] DO
B466      BEGIN
B466       WRITELN('NUME:',NUME);
B481       WRITELN('ADR:',ADR);
B49F       WRITELN('NR:',NR);
B4C1       WRITELN('NOTA:',NOTA:5:2);
B4F0       WRITELN;
B4F3      END;
B4F3    END;
B4F6  END{#P}.
```

```
1 CORICI     MARASTI   12  6.99
2 MANZ       RESITA    59  9.33
3 SERBAN     HEBE      38  7.01
4 SIMULESCU  PLAVAT    98  6.13

ELEVII CU NOTA MAI MARE DE 7
ELEVUL: 2

NUME:MANZ
ADR:RESITA
NR:59
NOTA: 9.33

ELEVUL: 3

NUME:SERBAN
ADR:HEBE
NR:38
NOTA: 7.01
```

P.IV.12.

4.2.3. Tipul mulţime

Tipul *mulţime* este un tip structurat care se defineşte în raport cu un tip de bază care este un tip *ordinal*. Dîndu-se tipul de bază, tipul mulţime se defineşte ca fiind mulţimea *tuturor submulţimilor* tipului de bază, inclusiv mulţimea vidă.

Fiecare valoare a tipului mulţimii este o mulţime ale cărei componente sînt valori distincte ale tipului de bază. Numărul maxim de valori pentru tipul de bază diferă de la o implementare la alta. In cazul lui HP4TM, acest număr este 256.

Specificarea unui tip mulţime se face conform diagramei din fig.4.17.

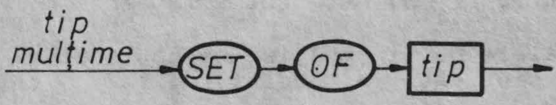


fig.4.17.

unde:

-*tip* este tipul de bază.

Ca şi în cazul celorlalte tipuri, acest tip nou definit poate rămîne anonim sau poate primi un nume într-o declaraţie de tip conform diagramei de sintaxă din fig.4.18.

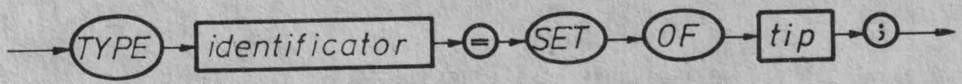


fig.4.18.

Mulţimile cu care lucrăm în program se pot alcătui din componentele lor folosind *constructori*. Un constructor are sintaxa din fig.4.19.

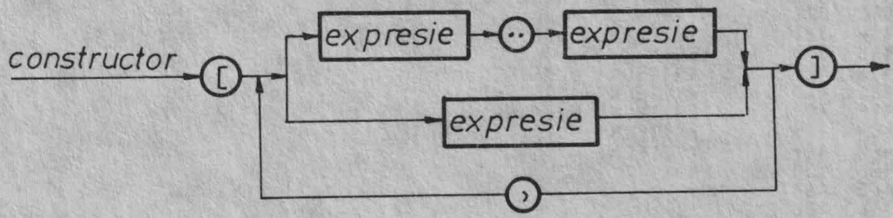


fig.4.19.

Cu alte cuvinte, un constructor determină o enumerare a elementelor unei mulţimi, cuprinsă între paranteze drepte.

Exemplu:

```
      VAR A:SET OF 0..20;
          B:SET OF 'A'..'Z';
```

Variabilelor de tip mulţime declarate li se pot atribui mulţimi prin următorii constructori:

```
      1) A:=[1,5..10];
         B:=['C'..'J'];

      2) A:=[1,5,10];
         B:=['A','C','D'];

      3) A:=[];
         B:=['C'..'A'];
```

In exemplul 1) A este o mulţime formată din elementele 1,5,6,7,9,10 iar B o mulţime formată din caracterele C,D,E,F,G,H,I,J. In exemplul 3) mulţimile A şi B desemnează mulţimea *vidă* deoarece atît constructorul [] cît şi un constructor de tipul [a..b] cu b>a desemnează o mulţime fără elemente.

Tipul de bază al unei mulţimi desemnate de un constructor este considerat cel mai "sărac" tip căruia îi aparţin valorile elementelor cuprinse între parantezele drepte. De exemplu, pentru constructorul [1,2,3] tipul de bază este considerat 1..3, pentru [1..3] este tot 1..3 etc.

Cu variabilele de tip mulţime se pot efectua calcule. Operatorii folosiţi sînt daţi în tabelul T.4.12.

Observaţie:

Operanzii trebuie să fie de acelaşi tip de bază ordinal, un tip de bază subdomeniu al celuilalt sau ambele tipuri de bază subdomenii ale aceluiaşi tip.

Tabel T.4.12.

Operator	Tip operanzi	Tip rezultat	Semnificaţie
+	Mulţimi	Mulţime	Reuniune
*	Mulţimi	Mulţime	Intersecţie
-	Mulţimi	Mulţime	Diferenţă
=	Mulţimi	Boolean	...este egală cu ...
<>	Mulţimi	Boolean	...este diferită de...
<=	Mulţimi	Boolean	...este inclusă în...
>=	Mulţimi	Boolean	...include pe...
IN	Ordinal,mulţime	Boolean	...aparţine lui...

Programul **REUNIUNE** (P.IV.13) determină reuniunea a două mulţimi de numere întregi.

```
AF2C   PROGRAM REUNIUNE;
AF2C   {EXEMPLU PENTRU TIPUL MULTIME}
AF2C   TYPE NUMERE=0..20;
AF2C   VAR E,N,M,J,I:INTEGER;
AF35       A,B,C:SET OF NUMERE;
AF35   BEGIN
AF3E   A:=[];
AF57   B:=[];
AF70    WRITELN('NUMARUL DE ELEMENTE DIN MULTIMEA A:');
AFA1    READ(J);
AFA7    WRITELN('NUMARUL DE ELEMENTE AL MULTIMII B:');
AFD7    READ(M);
AFDD    WRITELN('ELEMENTELE DIN A');
AFFB    FOR I:=1 TO J DO
B019     BEGIN
B01C      READ(E);
B022      A:=A+[E];
B061     END;
B065    WRITELN('ELEMENTELE DIN B');
B083    FOR I:=1 TO M DO
B0A1     BEGIN
B0A4      READ(E);
B0AA      B:=B+[E];
B0E9     END;
B0ED    C:=A + B;
B12F    FOR I:=0 TO 20 DO
B149     IF I IN C THEN WRITE(I:3);
B188   END {$P}.

ELEMENTELE DIN A
{ 5  4  7  8  3 }

ELEMENTELE DIN B
{ 2  3 10  6 11 14  7 }

ELEMENTELE MULTIMII C
{ 2  3  4  5  6  7  8 10 11 14 }
```

P.IV.13.

Programul **SIRURI** (P.IV.14) construieşte, pornind de la un şir de numere întregi două şiruri: unul cu elementele pare, celălalt cu cele impare. In acest scop am construit o mulţime ale cărei elemente sînt cifre pare. In cadrul programului se localizează ultima cifră a unui element din şir şi se cercetează apartenenţa sa la mulţimea cifrelor pare. In caz afirmativ elementul se depune în şirul cu numerele pare iar în caz contrar în cel cu numerele impare.

```
AEBC  PROGRAM SIRURI;
AEBC  TYPE TIP=SET OF 0..9;
AEBC       SIR=ARRAY[1..100] OF INTEGER;
AEBC  VAR NUMERE,SPAR,SIMPAR,SIRUL:SIR;
AEC5      I,J,K,N:INTEGER;
AEC5      PARE,IMPARE:TIP;
AEC5  BEGIN
AECE   PARE:=[0,2,4,6,8];
AF14   IMPARE:=[1,3,5,7,9];
AF5A   WRITELN('NUMARUL DE ELEMENTE AL SIRULUI: ');
AF88   READ(N);
AF8E   WRITELN;WRITELN('ELEMENTELE SIRULUI:');WRITELN;
AFB5   K:=0;J:=0;
AFC1   FOR I:=1 TO N DO
AFDF    BEGIN
AFE2     READ( SIRUL[I]);
B005     IF SIRUL[I] MOD 10 IN PARE THEN
B054         BEGIN
B054          J:=J+1;
B05B          SPAR[J]:=SIRUL[I]
B091         END
B099     ELSE
B09C         BEGIN
B09C          K:=K+1;
B0A3          SIMPAR[K]:=SIRUL[I]
B0D9         END
B0E1    END;
B0E5   WRITELN;
B0E8   WRITELN('SIRUL CU ELEMENTELE PARE:',CHR(13));
B116   FOR I:=1 TO J DO
B134    WRITELN('SPAR(',I,')=',SPAR[I]);
B188   WRITELN;
B18B   WRITELN('SIRUL CU ELEMENTELE IMPARE:',CHR(13));
B1BB   FOR I:=1 TO K DO
B1D9    WRITELN('SIMPAR(',I,')=',SIMPAR[I]);
B22F  END {$P}.
```

```
ELEMENTELE SIRULUI:

5 6 3 9 1 2 4 6 4 3 8 2

SIRUL CU ELEMENTELE PARE:

SPAR(1 )=6
SPAR(2 )=2
SPAR(3 )=4
SPAR(4 )=6
SPAR(5 )=4
SPAR(6 )=8
SPAR(7 )=2

SIRUL CU ELEMENTELE IMPARE:

SIMPAR(1 )=5
SIMPAR(2 )=3
SIMPAR(3 )=9
SIMPAR(4 )=1
SIMPAR(5 )=3
```

P. IV. 14.

PROBLEME PROPUSE

1. Se dă un tablou unidimensional A cu elemente numere reale. Să se scrie un program PASCAL pentru interschimbarea elementelor egal depărtate între ele.

2. Fiind dat un tablou bidimensional cu elemente numere reale, să se scrie programul PASCAL care calculează media aritmetică a elementelor strict pozitive aflate deasupra diagonalei principale.

3. Fiind date tablourile unidimensionale A şi B, fiecare avînd 20 de elemente caractere, să se scrie programul PASCAL care construieşte tabloul C obţinut din tablourile A şi B astfel: primul element din A, al doilea element din B etc...

4. Fiind date tablourile unidimensionale de la problema precedentă să se construiască mulţimea C avînd ca elemente caracterele din tablourile A şi B.

5. Să se scrie programul PASCAL care citeşte un articol cu următoarea structură: nume (20 caractere), adresa, nr.telefon, data naşterii:an, lună, zi. Să se afişeze persoanele care sînt născute în luna martie între anii 1950 şi 1980.

6. Să se scrie programul PASCAL de înmulţire a două polinoame. Coeficienţii celor două polinoame sînt daţi sub formă de tablou unidimensional.

7. Dîndu-se un tablou bidimensional de numere reale, să se reordoneze liniile tabloului în ordine descrescătoare a elementelor de pe prima coloană.

8. Dîndu-se un tablou unidimensional de elemente numere reale, să se scrie programul PASCAL care elimină elementele nule din tablou.

9. Dîndu-se patru mulţimi de elemente caractere, să se scrie programul PASCAL care calculează şi afişează următoarele rezultate:

A U B
A - B
A U (C - D)

10. Dîndu-se o matrice pătrată cu elemente caractere, să se scrie un program PASCAL care roteşte această matrice cu 90^o respectiv 180^o spre stînga şi spre dreapta şi obţine imaginea ei în oglindă.

V. SUBPROGRAME

Scrierea programului se face mult mai uşor dacă împărţim problema în subprobleme relativ independente, pentru fiecare din ele scriindu-se programe mult mai simple. De altfel, realizarea unui program de complexitate mare impune aproape ca o necesitate organizarea unor date şi a acţiunilor la care acestea trebuie supuse, sub formă de subprograme. In PASCAL există un mecanism evoluat de declarare şi utilizare a subprogramelor.

Subprogramele limbajului PASCAL sînt de două tipuri: *proceduri* şi *funcţii*. Diferenţa dintre ele constă în numărul valorilor calculate şi returnate programului apelant: procedura transmite *oricîte* astfel de valori, pe cînd funcţia transmite *o singură* valoare, acest lucru permiţînd ca apelul ei să se facă chiar din expresia care conţine valoarea calculată.

Atît procedurile, cît şi funcţiile, pot fi de două tipuri: *standard* (predefinite) şi *nestandard* (declarate în program). Procedurile şi funcţiile nestandard trebuie în mod obligatoriu declarate *înainte de a fi apelate*. In cazul general, un program PASCAL este format dintr-un program principal şi dintr-un număr oarecare de proceduri şi funcţii, apelabile din programul principal, sau unele din altele.

5.1. DOMENIUL DE VALABILITATE AL IDENTIFICATORILOR SI ETICHETELOR

Declararea procedurilor în cadrul altor proceduri, fiecare din acestea cu propriile declaraţii şi definiţii ridică o problemă deosebită: domeniul de valabilitate al identificatorilor şi etichetelor.

Prin *domeniu de valabilitate* înţelegem zona din program în care este cunoscută declaraţia sau definiţia unui identificator sau a unei etichete.

In PASCAL, la baza stabilirii domeniului de valabilitate stă conceptul de *bloc*. Conform diagramei din cap I, fig.1.3, blocul este corpul unui subprogram format din declaraţii, definiţii şi o instrucţiune compusă. Blocul are o proprietate importantă: toţi identificatorii şi etichetele definite în el sînt *locale* blocului şi cunoscute doar în interiorul blocului. Se defineşte astfel *structura de bloc*. Intr-un limbaj de programare cu structură de bloc, diferitele entităţi definite într-un bloc se "nasc" la intrarea în execuţia (activarea) blocului, apoi "dispar" la ieşirea din execuţia (dezactivarea) blocului respectiv. In acest fel se

obține o folosire mai bună a spațiului de memorare a datelor. De asemenea, există în acest fel posibilitatea activării recursive a unui bloc.

Atunci cînd fluxul de control intră într-un bloc, toate entitățile declarate în blocul respectiv sînt alocate pe *stiva de execuție*; la părăsirea unui bloc zona de pe stivă alocată blocului este eliberată astfel încît valorile aflate acolo nu mai pot fi referite.

O consecință a structurii de bloc este că procedurile și funcțiile pot fi *recursive*, adică se pot apela pe ele însele, fie direct, fie indirect printr-un lanț de apeluri. Fiecare invocare a unei proceduri determină alocarea de spațiu pe stivă pentru entitățile sale locale care sînt astfel distincte de entitățile corespunzătoare alocate în timpul altor apeluri ale aceleiași proceduri. Acest spațiu este eliberat la terminarea invocării respective.

In concluzie:

Domeniul de valabilitate al unui identificator sau etichete este tot textul blocului în care aceștia au fost definiți, inclusiv textele blocurilor subprogramelor declarate în bloc, cu excepția celor care redefinesc identificatorul sau eticheta. Intr-un punct al programului situat în blocul *B* sînt cunoscuți toți identificatorii definiți în *B* și în blocurile care înconjoară textual pe *B*. Pentru oricare dintre acești identificatori, definiția valabilă în punctul considerat este cea aflată în blocul cel mai apropiat de *B*. Același lucru este valabil și pentru etichete.

5.2. DEZVOLTAREA PROGRAMELOR PASCAL

In PASCAL există două posibilități de dezvoltare a programelor.

A) In *programarea ascendentă* subprogramele se declară în secțiunea de declarații a programului unul după altul, respectînd o ordine anume impusă de eventualele apeluri dintre acestea. Restricția se referă la regula generală privind *referirea identificatorilor numai după declararea lor*. In acest context este clar că apelul unei proceduri *p* într-o linie program este acceptat de compilator numai dacă procedura *p* a fost declarată într-o linie anterioară.

Un program scris în stilul ascendent va avea structura din fig. 5.1.

Observații:

1)-prin *antet procedură* înțelegem linia program în care se declară numele subprogramului; sintaxa ei se va prezenta detaliat in 5.3.1.

2)-într-un astfel de program, blocurile sînt independente;

3)-apelurile posibile într-un program cu structura de mai sus sînt:

* din programul principal se pot apela subprogramele notate *procedură-i*, pentru i=1,n în orice ordine;

* din *procedură-i* se pot apela subprogramele *procedură-k*, pentru k=1,i-1 în orice ordine.

```
antet;
 d  ┌               ┌ constante
 e  │ declaraţii    │ etichete
 c  │  globale      │ tipuri
 l  │               └ variabile
 a  │
 r  │               ┌ antet procedură-1
 a  │ declaraţii    │
 ţ  │               │   declaraţii locale
 i  │               │
 i  │               │   secţiune algoritmică procedură-1
    │ proceduri     │
 p  │               │ antet procedură-2
 r  │               │
 o  │    şi         │   declaraţii locale
 g  │               │
 r  │               │   secţiune algoritmică procedură-2
 a  │  funcţii      │
 m  │               │ .......................
    │               │
    │               │ antet procedură-n
    │               │
    │               │   declaraţii locale
    │               │
    └               └   secţiune algoritmică procedură-n

  secţiune algoritmică
  program principal
```

fig. 5. 1.

Simplificînd schema din fig.5.1., structura programului este reprezentată în fig.5.2.



fig. 5. 2.

B) In programarea *descendentă* subprogramele se declară *imbricate* unul în celălalt. Avînd în vedere că dintr-un bloc sînt vizibile (deci apelabile) numai acele subprograme care sînt declarate în secţiunea de declaraţii ataşate blocului respectiv, nu orice subprogram va fi apelabil dintr-un anume bloc, chiar dacă acesta este declarat într-o linie anterioară.

Un program scris în stilul descendent va avea structura din fig.5.3.

Observaţii:

1)- apelurile posibile într-un program cu structura din fig.5.3. sînt:

* din programul principal se poate apela subprogramul *procedură-1*;
* din *procedură-i* se poate apela *procedură-i+1*;

2)- comparînd cele două structuri, observăm că vom prefera dezvoltarea programului *ascendent* în cazul în care dorim să apelăm din programul principal orice subprogram declarat;

3)- în cazul programării *descendente*, avînd în vedere regulile după care se stabileşte domeniul de valabilitate al

identificatorilor, în general, se reduce numărul parametrilor şi al declaraţiilor locale;

4)- evident, în practică se pot combina cele două stiluri (*ascendent* şi *descendent*).

```
antet;
 d        declaraţii  { constante
 e        globale     { etichete
 c                    { tipuri
 l                    { variabile
 a
 r                    { antet procedură-1
 a
 ţ        declaraţii     declaraţii   declaraţii
 i                       proprii            locale
 i                       procedură-1 { antet procedură-2
          proceduri
 p                                      declaraţii  declaraţii
 r                                      proprii         locale
 o           şi                         procedură-2 ............
 g
 r                                           { antet procedură-n
 a          funcţii
 m                                                 declaraţii
                                                       locale

                                                   secţiune
                                                   algoritmică
                                                   procedură-n
                                        ........................
                                        secţiune
                                        algoritmică
                                        procedură-2
                         secţiune
                         algoritmică
                         procedură-1

 secţiune algoritmică
 program principal
```

fig. 5. 3.

Schematic, structura programului este reprezentată în fig.5.4.



fig. 5. 4.

5.3. PROCEDURI

5.3.1. Declararea procedurilor. Parametri formali, parametri efectivi

Declaraţiile de proceduri reprezintă părţi distincte ale programului PASCAL. Aceste părţi pot fi identificate în program conform declaraţiei de sintaxă din fig.5.5.

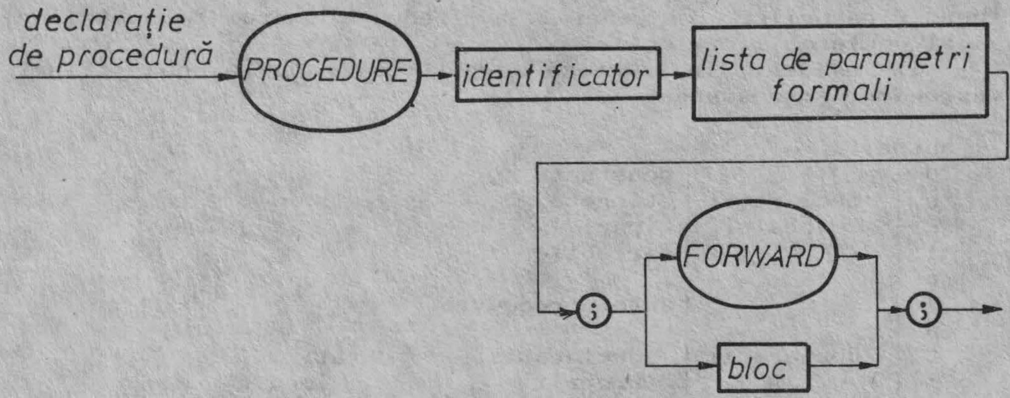


fig. 5.5.

Lista de *parametri formali* este formată din mai multe subliste conform diagramei din fig.5.6.

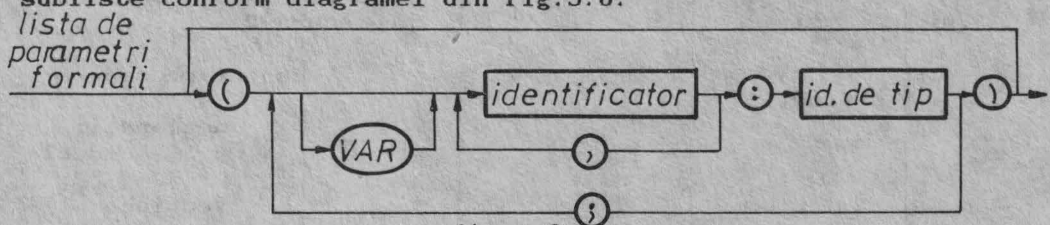


fig. 5.6.

Se observă că sublistele se separă între ele prin ';', iar parametrii în cadrul sublistei se despart prin ',' In PASCAL există următoarele reguli de utilizare a parametrilor formali:

- un parametru formal apare ca un identificator; acestuia i se asociază un tip, specificat prin *id. de tip*;
- parametrii formali declarați într-o procedură sînt cunoscuți în întreg textul procedurii și necunoscuți în afară;
- în cadrul procedurii, parametrii formali se manifestă ca niște varaibile de tipul respectiv;
- dacă lista parametrilor formali este vidă, atunci sau nu există schimb de informații cu restul programului sau schimbul se execută prin intermediul variabilelor globale.

In limbajul PASCAL se pot defini trei mari categorii de parametri formali:

a) *-parametri transmiși prin valoare*: sublista respectivă nu este precedată de nici un cuvînt cheie (VAR, FUNCTION, PROCEDURE); acești parametri pot fi modificați în corpul procedurii dar valorile noi *nu se transmit* către blocul apelant, ei sînt doar *parametri de intrare*;

b) *-parametri transmiși prin referință*: sublista respectivă este precedată de cuvîntul cheie VAR, iar parametrii actuali sînt *variabile*; acești parametri pot fi modificați în corpul procedurii, ei sînt *parametri de ieșire*, nu sînt transmiși prin valoare ci se transmite adresa lor, deci în blocul apelant valorile parametrilor actuali vor fi modificate;

c) *-parametri funcții sau proceduri*: sublista respectivă este precedată de cuvintele cheie FUNCTION sau PROCEDURE; avînd în vedere că în implementarea aleasă nu pot exista parametri funcții sau proceduri, nu vom trata acest caz.

Activarea unei proceduri se realizează prin *instrucţiunea de apel (instrucţiune procedurală)* care constă în identificatorul procedurii, urmat eventual de o listă de *parametri actuali (efectivi)*.

Referitor la utilizarea listelor de parametri (formali şi efectivi), în PASCAL există următoarele reguli:

a)-între lista parametrilor formali şi lista parametrilor efectivi există o corespondenţă *unu-la-unu* determinată de ordinea parametrilor în listă;

b)-în cazul transmiterii prin valoare, parametrii efectivi sînt expresii; tipul unei astfel de expresii trebuie să fie compatibil cu tipul parametrului formal corespunzător; din punct de vedere al compatibilităţii trebuie respectate regulile stabilite la atribuire între expresie şi variabilă;

c)-în cazul transmiterii prin referinţă, parametrul efectiv asociat cu un parametru-variabilă trebuie să fie o variabilă de acelaşi tip cu parametrul formal; orice operaţie efectuată asupra unui parametru-variabilă va fi realizată în fond asupra parametrului efectiv asociat.

5.3.2. Proceduri standard de intrare/ieşire

In PASCAL există o serie de proceduri standard, care nu trebuie declarate, ele asigurînd o serie de facilităţi necesare în orice program.

Procedura WRITE

-este folosită pentru a transmite date către monitor sau către imprimantă.

Forma generală a apelului procedurii este:

```
WRITE(p1,p2,...,pn);
```

şi este echivalentă cu:

```
BEGIN
  WRITE(p1);
  WRITE(p2);
  ..........
  WRITE(pn)
END;
```

Parametrii *p1,p2,...,pn* pot avea una din următoarele forme:

```
e
e:m
e:m:n
e:m:H
```

unde *e* este expresia de tipărit, *m*, *n* sînt expresii întregi, iar *H* o constantă literală indicînd tipărirea în hexa.

Avînd în vedere forma pe care o pot avea parametrii, precum şi tipurile pe care le pot avea expresiile, semnalăm 5 cazuri:

1)-*e* este de tip **INTEGER** şi se foloseşte una din formele e sau e:m;

In acest caz valoarea expresiei este convertită într-un şir de caractere, urmat de un spaţiu. Lungimea şirului

poate fi mărită (cu mai multe spaţii în faţă) prin folosirea lui *m*, care specifică numărul total de caractere transmise spre ieşire. Dacă *m* este mai mic decît numărul de cifre al expresiei *e*, atunci *m* nu se ia în considerare (se consideră absent) şi se transmite spre ieşire valoarea expresiei *e*, fără a se tipări şi spaţiul la sfîrşit.

2)-*e* este de tip **INTEGER** şi se foloseşte forma *e*:*m*:*H*;

In acest caz, înaintea transmiterii valorii lui *e* se realizează o conversie în hexazecimal:

-dacă *m*=1 sau *m*=2 atunci este transmisă valoarea ultimelor *m* cifre hexa ale expresiei pe *m* poziţii;

-dacă *m*=3 sau *m*=4 este transmisă valoarea completă a lui *e*, în hexazecimal, pe 4 poziţii;

-dacă *m*>4, vor fi inserate spaţii în faţa valorii complete a lui *e*, în hexazecimal, pînă la completarea celor *m* caractere. Dacă este cazul, sînt inserate zerouri la stînga, iar numerele negative sînt complementate. Ilustrarea celor spuse mai sus se poate urmări în programele **PV1** şi **TEST** (P.V.1).

```
AC80   PROGRAM PV1;
AC80   VAR M,N:INTEGER;
AC89       E:INTEGER;
AC89   BEGIN
AC92    E:=1025;
AC98    FOR M:= 1 TO 10 DO
ACB2     BEGIN
ACB5      WRITE('m=',M:2,' E=',E:M:H);
ACE7      WRITE(CHR(13))
ACEE     END;
ACF1   END {$P}.

m= 1 E=1
m= 2 E=01
m= 3 E=0401
m= 4 E=0401
m= 5 E= 0401
m= 6 E=  0401
m= 7 E=   0401
m= 8 E=    0401
m= 9 E=     0401
m=10 E=      0401
```

```
AC82   PROGRAM TEST;
AC82   VAR M,N:INTEGER;
AC8B       E:INTEGER;
AC8B   BEGIN
AC94    E:=-1025;
AC9D    FOR M:= 1 TO 10 DO
ACB7     BEGIN
ACBA      WRITE('m=',M:2,' E=',E:M:H);
ACEC      WRITE(CHR(13))
ACF3     END;
ACF6   END {$P}.

m= 1 E=F
m= 2 E=FF
m= 3 E=FBFF
m= 4 E=FBFF
m= 5 E= FBFF
m= 6 E=  FBFF
m= 7 E=   FBFF
m= 8 E=    FBFF
m= 9 E=     FBFF
m=10 E=      FBFF
```

P.V.1.

3)-*e* este de tip **REAL**; se poate utiliza una din formele e, e:m, e:m:n;

Valoarea lui *e* este convertită într-un şir de caractere care reprezintă un număr real. Formatul reprezentării este determinat de *n*:

-dacă *n* nu este prezent, atunci numărul este transmis în notaţie cu exponent. Dacă numărul este negativ, se transmite un semn - (minus) în faţa mantisei; în caz contrar este transmis un spaţiu; numărul este transmis întotdeauna cu cel puţin una şi cel mult 5 zecimale, iar exponentul este transmis întotdeauna cu semn (plus sau minus); aceasta înseamnă că lungimea maximă a acestei reprezentări este de 8 caractere;

-dacă *m* este mai mic decît 8 (ex. *m*=7), atunci va fi transmisă întotdeauna lungimea completă de 12 caractere;

-dacă *m*>=8, vor fi transmise una sau mai multe zecimale (pînă la 5 zecimale, cînd *m*=12);

-dacă *m*>12, în faţa numărului vor fi inserate spaţii.

Ilustrarea celor prezentate mai sus se poate urmări în programul **TEST** (P.V.2), pe următoarele exemple:

e = -1.025E10
e = 100000/3
e = 1/333

```
AC81   PROGRAM TEST;
AC81   VAR M,N:INTEGER;
AC8A       E:REAL;
AC8A   BEGIN
AC93    E:=-1.025E10;
ACA8    FOR M:= 7 TO 15 DO
ACC2     BEGIN
ACC5      WRITE('m=',M:2,' E=',E:M);
ACFE      WRITE(CHR(13))
AD05     END;
AD08   END {$P}.

m= 7 E=-1.02500E+10
m= 8 E=-1.0E+10
m= 9 E=-1.02E+10
m=10 E=-1.025E+10
m=11 E=-1.0250E+10
m=12 E=-1.02500E+10
m=13 E= -1.02500E+10
m=14 E=  -1.02500E+10
m=15 E=   -1.02500E+10

m= 7 E= 3.33333E+03
m= 8 E= 3.3E+03
m= 9 E= 3.33E+03
m=10 E= 3.333E+03
m=11 E= 3.3333E+03
m=12 E= 3.33333E+03
m=13 E=  3.33333E+03
m=14 E=   3.33333E+03
m=15 E=    3.33333E+03

m= 7 E= 3.00300E-03
m= 8 E= 3.0E-03
m= 9 E= 3.00E-03
m=10 E= 3.003E-03
m=11 E= 3.0030E-03
m=12 E= 3.00300E-03
m=13 E=  3.00300E-03
m=14 E=   3.00300E-03
m=15 E=    3.00300E-03
```

P.V.2.

Dacă se foloseşte forma *e*:*m*:*n*, atunci expresia *e* va fi scrisă în reprezentare cu punct fix, cu *n* zecimale. Cînd lungimea *m* a cîmpului este suficient de mare, vor fi transmise în faţă spaţii. Dacă *n*=0, *e* va fi transmis ca întreg. Dacă *e* este prea mare pentru a fi scris în lungimea *m* specificată de cîmp, atunci el va fi transmis în formatul cu exponent, pe un cîmp de lungime *m*, ca în cazul anterior. Marca zecimală '.' se ia în considerare la stabilirea numărului de caractere ce trebuie transmise. Exemplele din programul **TEST** (P.V.3) ilustrează cele prezentate mai sus, pentru:

e = 23.455
e = -23.455

```
ACB6   10 PROGRAM TEST;
ACB6   20 VAR M,N:INTEGER;
ACBF   30     E:REAL;
ACBF   40 BEGIN
ACC8   50  E:=23.455;
ACD5   60  FOR M:= 4 TO 8 DO
ACEF   70   BEGIN
ACF2   80    FOR N:=0 TO 4 DO
AD0C   90     BEGIN
AD0F  100      WRITE('m=',M:1,' n=',N:1,' E=',E:M:N);
AD64  110      WRITE(CHR(13))
AD6B  120     END;
AD6E  130    WRITE(CHR(13))
AD75  140   END;
AD78  150 END ($P).
```

```
m=4 n=0 E=  23
m=4 n=1 E=23.5
m=4 n=2 E= 2.34550E+01
m=4 n=3 E= 2.34550E+01
m=4 n=4 E= 2.34550E+01

m=5 n=0 E=   23
m=5 n=1 E= 23.5
m=5 n=2 E=23.45
m=5 n=3 E= 2.34550E+01
m=5 n=4 E= 2.34550E+01

m=6 n=0 E=    23
m=6 n=1 E=  23.5
m=6 n=2 E= 23.45
m=6 n=3 E=23.455
m=6 n=4 E= 2.34550E+01

m=7 n=0 E=     23
m=7 n=1 E=   23.5
m=7 n=2 E=  23.45
m=7 n=3 E= 23.455
m=7 n=4 E=23.4550

m=8 n=0 E=      23
m=8 n=1 E=    23.5
m=8 n=2 E=   23.45
m=8 n=3 E=  23.455
m=8 n=4 E= 23.4550


m=4 n=0 E= -23
m=4 n=1 E=-2.34550E+01
m=4 n=2 E=-2.34550E+01
m=4 n=3 E=-2.34550E+01
m=4 n=4 E=-2.34550E+01

m=5 n=0 E=  -23
m=5 n=1 E=-23.5
m=5 n=2 E=-2.34550E+01
m=5 n=3 E=-2.34550E+01
m=5 n=4 E=-2.34550E+01
```

```
m=6 n=0 E=   -23
m=6 n=1 E= -23.5
m=6 n=2 E=-23.45
m=6 n=3 E=-2.34550E+01
m=6 n=4 E=-2.34550E+01

m=7 n=0 E=    -23
m=7 n=1 E=  -23.5
m=7 n=2 E= -23.45
m=7 n=3 E=-23.455
m=7 n=4 E=-2.34550E+01

m=8 n=0 E=     -23
m=8 n=1 E=   -23.5
m=8 n=2 E=  -23.45
m=8 n=3 E= -23.455
m=8 n=4 E=-23.4550
```

P.V.3.

4)-*e* este de tip **CHAR** sau de tip **şir de caractere**;

Se poate folosi una din formele *e* sau *e*:*m*. Caracterul sau şirul de caractere va fi transmis pe o lungime minimă de cîmp, egală cu 1 (pentru caractere), sau egală cu lungimea şirului (în cazul şirurilor). Dacă *m* este suficient de mare vor fi inserate spaţii la stînga.

Cînd expresia care urmează să fie scrisă este simplă, de tip caracter, atunci WRITE(e) transmite către monitor sau imprimantă, după caz, valoarea pe 8 biţi a expresiei *e*. In PASCAL HP4TM există cîteva coduri care într-o procedură WRITE au un rol aparte, fiind considerate coduri de control. Aceste coduri sînt:

CHR(8) -mută cursorul înapoi cu un pas pe ecran, cu ştergere (BACK SPACE);

CHR(12)-şterge ecranul sau execută salt la pagină nouă, dacă este transmis la imprimantă;

CHR(13)-execută <CR> şi <LF> (Carriage Return şi Line Feed) adică termină linia şi execută salt la linie nouă;

CHR(16)-comută ieşirea de la monitor la imprimantă şi invers.

Observaţii:

1)-înainte de terminarea programului trebuie să ne asigurăm că ieşirea este pe monitor, pentru a putea continua lucrul;

2)-cînd se doreşte lucrul alternativ cu monitorul şi imprimanta, citirea datelor se face de la tastatură prin vizualizare pe monitor iar scrierea se face la imprimantă, deci trebuie să avem grijă ca în momentul citirii datelor canalul de ieşire spre monitor să fie deschis, iar în momentul scrierii la imprimantă canalul de ieşire spre imprimantă să fie deschis;

3)-la pornire, HP4TM are conectată ieşirea spre monitor.

```
AD29   10 PROGRAM TEST;
AD29   20 VAR M:INTEGER;
AD32   30     E:ARRAY[1..8] OF CHAR;
AD32   40     CAR:CHAR;
AD32   50 BEGIN
AD3B   60  CAR:='*';
AD40   70  E:='EX EMPLU';
AD58   80  WRITE('CAR=',CAR,CHR(13),'E=',E,CHR(13),CHR(13));
AD97   90  FOR M:= 0 TO 4 DO
ADB1  100   BEGIN
ADB4  110    WRITE('m=',M:1,' CAR=',CAR:M);
ADED  120    WRITE(CHR(13))
ADF4  130   END;
ADF7  140  FOR M:= 6 TO 12 DO
AE11  150   BEGIN
AE14  160    WRITE('m=',M:2,' E=',E:M);
```

```
AE4D  170     WRITE(CHR(13))
AE54  180    END;
AE57  190 END {$P}.
```

```
CAR=*
E=EX EMPLU

m=0 CAR=*
m=1 CAR=*
m=2 CAR= *
m=3 CAR=  *
m=4 CAR=   *
m= 6 E=EX EMPLU
m= 7 E=EX EMPLU
m= 8 E=EX EMPLU
m= 9 E= EX EMPLU
m=10 E=  EX EMPLU
m=11 E=   EX EMPLU
m=12 E=    EX EMPLU
```

P.V.4.

5)-*e* este de tip **BOOLEAN**; pot fi utilizate formele e sau e:m;

In ambele cazuri se transmite 'TRUE' sau 'FALSE', în funcţie de valoarea lui *e*, folosind o lungime minimă de cîmp de 4 sau 5 caractere. Programele **NPRIM** (P.V.5a şi P.V.5b) stabilesc dacă un număr natural introdus de la tastatură este sau nu prim, răspunsul la întrebare -'TRUE' sau 'FALSE'- fiind atribuit variabilei booleene PRIM.

Se observă (în P.V.5a) modul de scriere a celor două valori 'TRUE' şi 'FALSE' pe 4/5 caractere şi faptul că listarea este "urîtă". Prin simpla scriere cu format (liniile 170 şi 180 în P.V.5b) listarea se face ordonat, tabulat.

```
AD10   10 PROGRAM NPRIM;
AD10   20 VAR I,N:INTEGER;
AD19   30     PRIM:BOOLEAN;
AD19   40 BEGIN
AD22   50  REPEAT
AD22   60  WRITE ('N=');
AD32   70  READ(N);
AD38   80  I:=2;
AD3E   90  WHILE (I<N) AND ((N MOD I) <>0) DO I:=I+1;
AD78  100  PRIM:=I=N;
AD89  110  CASE PRIM OF
AD8C  120    TRUE:WRITELN(PRIM,' NUMARUL ',N:6,' ESTE PRIM');
ADD2  130    FALSE:WRITELN(PRIM,' NUMARUL ',N:6,' NU ESTE PRIM')
AE15  140  END;
AE18  150  UNTIL FALSE
AE1A  160 END {$P}.
```

```
TRUE NUMARUL     67 ESTE PRIM

TRUE NUMARUL   4567 ESTE PRIM

FALSE NUMARUL   7653 NU ESTE PRIM

TRUE NUMARUL      2 ESTE PRIM

TRUE NUMARUL      3 ESTE PRIM

FALSE NUMARUL      4 NU ESTE PRIM

FALSE NUMARUL  32767 NU ESTE PRIM

TRUE NUMARUL  15737 ESTE PRIM

FALSE NUMARUL    987 NU ESTE PRIM
```

P.V.5a.

```
AD14   10 PROGRAM NPRIM;
AD14   20 VAR I,N:INTEGER;
AD1D   30     PRIM:BOOLEAN;
AD1D   40 BEGIN
AD26   50  REPEAT
AD26   60  WRITE ('N=');
AD36   70  READ(N);
AD3C   80  I:=2;
AD42   90  WHILE (I<N) AND ((N MOD I) <>0) DO I:=I+1;
AD7C  100  PRIM:=I=N;
AD8D  110  CASE PRIM OF
AD90  120    TRUE:WRITELN(PRIM:6,' NUMARUL ',N:6,' ESTE PRIM');
ADE3  130    FALSE:WRITELN(PRIM:6,' NUMARUL ',N:6,' NU ESTE PRIM')
AE33  140  END;
AE36  150  UNTIL FALSE
AE38  160 END {$P}.

  TRUE NUMARUL     67 ESTE PRIM

  TRUE NUMARUL   4567 ESTE PRIM

 FALSE NUMARUL   7653 NU ESTE PRIM

  TRUE NUMARUL      2 ESTE PRIM

  TRUE NUMARUL      3 ESTE PRIM

 FALSE NUMARUL      4 NU ESTE PRIM

 FALSE NUMARUL  32767 NU ESTE PRIM

  TRUE NUMARUL  15737 ESTE PRIM

 FALSE NUMARUL    987 NU ESTE PRIM
```

P.V.5b.

Procedura WRITELN

-asemenea procedurii WRITE, este folosită pentru a transmite date către monitor sau către imprimantă.

Forma generală a apelului procedurii este:

WRITELN (p1,p2,...,pn);

Avînd acelaşi rol ca şi procedura WRITE, sînt valabile toate cazurile tratate anterior; singura deosebire constă în faptul că după terminarea afişării valorilor tuturor parametrilor se execută automat salt la linie nouă, deci:

WRITELN (p1,p2,...,pn);

este echivalent cu

WRITE (p1,p2,...pn,CHR(13));

precum şi cu

```
BEGIN
  WRITE (p1);
  WRITE (p2);
  ...........
  WRITE (pn);
  WRITE(CHR(13))
END;
```

WRITELN se poate apela şi fără parametri (WRITELN;), caz în care are ca efect doar salt la linie nouă.

Exemple se pot urmări în programul **WRITELNEX** (P.V.6).

```
AC5E   10 PROGRAM WRITELNEX;
AC5E   20 VAR I:INTEGER;
AC67   30 BEGIN
AC70   40  WRITELN;
AC73   50  FOR I:=1 TO 10 DO
AC8D   60   WRITELN(I:3,' ',I*I:4,' ',I*I*I:5)
ACD6   70 END {$P}.
```

```
  1   1    1
  2   4    8
  3   9   27
  4  16   64
  5  25  125
  6  36  216
  7  49  343
  8  64  512
  9  81  729
 10 100 1000
```

P.V.6.

Procedura PAGE

Forma generală a apelului procedurii este:

PAGE;

Această procedură este echivalentă cu WRITE (CHR(12)) şi are ca efect ştergerea ecranului dacă ieşirea este direcţionată spre monitor, sau saltul la pagină nouă dacă ieşirea este direcţionată spre imprimantă.

Observaţie:

Pentru scrierea din prima linie a ecranului se procedează în felul următor:

PAGE;
WRITE (CHR(22),CHR(0),CHR(0));

Remarcăm că deoarece CHR(22) reprezintă codul instrucţiunii BASIC "PRINT AT", se aşteaptă încă doi parametri: linia şi coloana (în cazul nostru 0, 0 deci CHR(0), CHR(0)).

Utilizarea procedurii PAGE este ilustrată în programul **WRITELNEX** (P.V.7).

```
AC88   10 PROGRAM WRITELNEX;
AC88   20 VAR I:INTEGER;
AC91   30 BEGIN
AC9A   40  PAGE;
AC9F   50  WRITE(CHR(22),CHR(0),CHR(0));
ACB4   60  WRITELN;
ACB7   70  FOR I:=1 TO 10 DO
ACD1   80   WRITELN(I:3,' ',I*I:4,' ',I*I*I:5)
AD1A   90 END {$P}.
```

P.V.7.

Procedura READ

-este folosită pentru a transmite date de la tastatură sau din fişierele utilizatorului (în cazul altor implementări decît HP4TM).

Forma generală a apelului procedurii este:

READ (v1,v2,...,vn);

Introducerea datelor se realizează prin intermediul unei zone tampon, deschisă în timpul executării programului,

zonă care inițial este goală, cu excepția (în cazul implementării HP4TM) a unui caracter *sfîrșit de linie* (EOL).

Se poate considera că accesul la această zonă are loc printr-o fereastră de text care permite vizualizarea în zonă a unui singur caracter odată. Dacă această fereastră de text este poziționată în dreptul unui caracter EOL și operația de citire este încheiată, va fi citită în zona tampon o nouă linie de text de la tastatură. Cind se citește această linie, sînt recunoscute diferite coduri de control specifice implementării HP4TM. Aceste coduri sînt:

```
RETURN (ENTER)      -sfîrșit de linie;
CC (CAPS SHIFT + 1) -revenire în editor (EDIT);
CH (CAPS SHIFT + 0) -șterge ultimul caracter introdus
                     (DELETE);
CI (CAPS SHIFT + 8) -deplasarea la următoarea poziție TAB;
CP (CAPS SHIFT + 3) -schimbă ieșirea spre imprimantă
                     (dacă există), iar dacă ieșirea este
                     la imprimantă, se revine la monitor;
CX (CAPS SHIFT + 5) -șterge întreaga linie.
```

Forma generală a apelului procedurii READ este echivalentă cu:

```
BEGIN
  READ(v1);
  READ(v2);
  .........
  READ(vn)
END;
```

Variabilele *v1,v2,...,vn* pot fi de tip caracter, șir de caractere, întreg sau real.

Ținînd cont de tipul variabilei *V*, READ(V) are efecte diferite; in acest sens *trebuie considerate 4 cazuri*:

1)-v este de tip caracter;

In acest caz, READ(V) citește un caracter din zona tampon de intrare și îl atribuie lui *V*. Dacă fereastra din tampon este poziționată pe un caracter *sfîrșit-de-linie* (CHR(13)), atunci funcția standard EOLN va primi valoarea TRUE și va fi citită o nouă linie de la tastatură. Cînd operația de citire este ulterior executată, fereastra de text va fi poziționată la începutul unei linii noi.

Este important de reținut că EOLN are valoarea TRUE la pornirea programului. Aceasta înseamnă că dacă la primul READ se citește o variabilă de tip caracter, se returnează o valoare CHR(13), *urmată de citirea unei noi linii* și deci următoarea citire de tip caracter va returna primul caracter din noua linie.

In programul **TESTREAD** (P.V.8), s-au introdus pe rînd caracterele 'TEST READ CU INLOCUIRE CHR(13)' și au fost scrise la imprimantă (s-a utilizat constanta CHR(16) pentru scrierea la imprimantă și revenirea imediată la monitor pentru continuarea citirii). Se observă că primul caracter transmis spre imprimantă a fost '#', deci primul caracter citit a fost CHR(13).

Programul conţine un ciclu infinit din cauza liniei 100 în care condiţia niciodată nu va fi îndeplinită (FALSE nu poate să devină TRUE). Pe de altă parte, citind astfel, nu avem obligaţia să precizăm numărul caracterelor, nici să semnalăm (cu CHR(13) sau un caracter anume) sfîrşitul citirii. Execuţia se întrerupe cu codul de control CC.

```
AC79   10 PROGRAM TESTREAD;
AC79   20 CONST P=CHR(16);
AC79   30 VAR CAR:CHAR;
AC82   40 BEGIN
AC8B   50  REPEAT
AC8B   60   READ(CAR);
AC94   70   IF CAR=CHR(13) THEN
ACA7   80      CAR:='#';
ACAC   90   WRITE(P,CAR,P)
ACBC  100  UNTIL FALSE
ACBE  110 END {$P}.

#TEST READ CU INLOCUIRE CHR(13)#
```

P.V.8.

2)-v este de tip şir de caractere;

Un şir de caractere poate fi citit cu READ şi, în acest caz, vor fi citite o serie de caractere, ori pînă se atinge numărul de caractere rezultat din definiţia şirului, ori pînă cînd EOLN=TRUE. Dacă şirul nu este completat prin citire (adică dacă *sfîrşit-de-linie* este atins înainte ca întregul şir să fie atribuit), atunci pînă la sfîrşitul şirului se completează cu caractere nule (CHR(0)); astfel, programatorul are posibilitatea să evalueze lungimea şirului care a fost citit. Observaţiile făcute la cazul 1) sînt valabile.

In programul **TESTREAD** (P.V.9) se citeşte cu READ(CHR13) caracterul iniţial *sfîrşit-de-linie* (CHR(13)), apoi cu READ(A) şirul A de tip caracter. La tastarea lui ENTER (CHR(13)), se întrerupe introducerea caracterelor, se tipăreşte şirul A (cu spaţii la sfîrşit - deoarece CHR(0) nu este tipăribil - pe monitor apare!) apoi parcurgîndu-se şirul se înlocuieşte CHR(0) cu *'#'*. La o nouă imprimare a lui A, în locul spaţiilor vor apare 7 caractere *'#'*. Se remarcă folosirea constantelor P şi L pentru comutarea ieşirii spre imprimantă (şi invers) respectiv pentru saltul la linie nouă.

```
ACCC   10 PROGRAM TESTREAD;
ACCC   20 CONST P=CHR(16);
ACCC   30       L=CHR(13);
ACCC   40 VAR A:ARRAY[1..15] OF CHAR;
ACD5   50     I:INTEGER;
ACD5   60     CHR13:CHAR;
ACD5   70 BEGIN
ACDE   80  READ(CHR13);
ACE4   90  READ(A);
ACEC  100  WRITE(P,A,L);
ACFE  110  FOR I:=1 TO 15 DO
AD18  120   IF A[I]=CHR(0) THEN
AD44  130      A[I]:='#';
AD64  140  WRITE(A,L,P)
AD76  150 END {$P}.

TEST SIR
TEST SIR#######
```

P.V.9.

Observaţie:

Citirea şirurilor de caractere diferă de la o implementare la alta.

3)-v este de tip întreg;

In acest caz, READ(V) citeşte un şir de caractere care reprezintă un întreg, aşa cum a fost acesta definit. Toate spaţiile şi caracterele *sfîrşit-de-linie* din stînga numărului sînt ignorate.

Dacă numărul citit nu aparţine intervalului [-MAXINT,MAXINT] va fi emis mesajul de eroare 'Number too large', iar execuţia programului va fi încheiată.

Dacă, după ce toate spaţiile şi caracterele *sfîrşit-de-linie* au fost ignorate, primul caracter nu este o cifră sau un semn ('+' sau '-'), va fi emis mesajul de eroare 'Number expected', iar execuţia programului va fi oprită

4)-v este de tip real;

In acest caz vor fi citite o serie de caractere reprezentînd un număr real, în acord cu sintaxa descrisă în capitolul II.

Ca şi în cazul anterior, toate spaţiile şi caracterele *sfîrşit-de-linie* din faţa primului caracter sînt ignorate şi, în mod analog, primul caracter *trebuie să fie o* cifră sau un semn. In caz contrar va fi emis mesajul de eroare 'Number expected'; dacă 'E' este prezent dar nu este urmat de o cifră sau un semn, este emis mesajul 'Exponent expected' şi execuţia programului este oprită.

Dacă numărul citit este prea mare sau prea mic, adică modulul lui nu aparţine intervalului [5.9E-39,3.4E38], va apare mesajul 'Overflow' şi execuţia programului este oprită.

Avînd în vedere că în PASCAL HP4TM cel mai mare întreg este MAXINT=32767, orice număr mai mare trebuie tratat ca număr real.

Lungimea mantisei unui număr real este de 23 de biţi. Din această cauză precizia obţinută prin utilizarea numerelor reale este de aproximativ 6 cifre semnificative. Remarcăm că precizia scade dacă rezultatul unui calcul este mult mai mic decît valorile absolute ale argumentelor sale.

Exemplu:

2.00002 - 2 NU va fi 0.00002 ci 0.0!

Din cauza modului în care se reprezintă numerele reale în memorie, nu are sens să se utilizeze mai mult de 7 cifre semnificative cînd se specifică mantisa unui număr real, deoarece cifrele în plus sînt ignorate. Cînd precizia este importantă, este bine să fie evitate zerourile semnificative de după marca zecimală, deoarece ele sînt numărate ca cifre semnificative.

Exemplu:

0.000123456 este mai puţin precis decît 1.23456E-4.

Dacă în program este necesară citirea mai multor valori, de diferite tipuri, acestea pot fi tastate în continuare (fără a tasta ENTER - *sfîrşit de linie*), dar separate între ele prin spaţii.

In programul **READ** (P.V.10), presupunem că în zona tampon avem şirul de caractere:

EXEMPLU*** 13.1 33 -10 0314.9 -15E-1 99.04EOL

Prima procedură READ(CAR) atribuie variabilei CAR, de tip caracter, caracterul *sfîrşit-de-linie* (EOL) existent în tampon în momentul lansării în execuţie; variabila SIR primeşte valoarea 'EXEMPLU***'; următoarele două spaţii sînt ignorate, deoarece urmează a fi citită o variabilă reală A, căreia i se atribuie valoarea 13,1. Celor trei elemente din şirul de întregi (SIRDEINTREGI) li se atribuie pe rînd valorile 33,-10 şi 314 (ultimul element este luat în considerare pînă la întîlnirea unui caracter care nu este cifră); variabilei CAR i se atribuie caracterul '.', iar celor două variabile reale B şi C valorile 9 respectiv -15E-1. Dacă următorul READ citeşte o variabilă reală, acesteia i se va atribui valoarea 99,04.

Rezultatele execuţiei programului **READ** (P.V.10) sînt separate prin caracterul '/' pentru o mai bună vizualizare a atribuirilor făcute prin citire.

```
AD51    10 PROGRAM READ;
AD51    20 VAR A,B,C:REAL;
AD5A    30     SIR:ARRAY[1..10] OF CHAR;
AD5A    40     I,INTREG:INTEGER;
AD5A    50     SIRDEINTREGI:ARRAY[1..3] OF INTEGER;
AD5A    60     CAR:CHAR;
AD5A    70 BEGIN
AD63    80  READ(CAR);
AD69    90  READ(SIR,A);
AD7B   100  FOR I:=1 TO 3 DO READ(SIRDEINTREGI[I]);
ADBE   110  READ(CAR,B,C);
ADD8   120  WRITE(CHR(16));
ADDF   130  WRITE(SIR,'/',A,'/');
ADFB   140  FOR I:=1 TO 3 DO WRITE(SIRDEINTREGI[I]);
AE3F   150  WRITE('/',CAR,'/',B,'/',C);
AE68   160  WRITE(CHR(16));
AE6F   170 END {$P}.

EXEMPLU***/ 1.31000E+01/33 -10 314 /./ 9.00000E+00/-1.50000E+00
```

P.V.10.

Procedura READLN

-asemenea procedurii READ, este folosită pentru a transmite date de la tastatură sau din fişierele utilizatorului;

Forma generală a apelului procedurii este:

READLN(v1,v2,...,vn);

Efectul este similar cu cel al procedurii standard READ, cu singura deosebire că *după* citirea valorilor variabilelor v1,v2,...,vn restul caracterelor din zona tampon pînă la caracterul CHR(13) se vor ignora, apoi se trece automat la începutul liniei următoare.

Efectul apelului procedurii standard READLN fără parametri (READLN;) este poziţionarea citirii la primul caracter ce urmează după caracterul CHR(13) din zona tampon.

READLN poate fi folosită pentru "a sări" peste linia vidă prezentă la începutul execuţiei programului, deci are efectul citirii într-o nouă zonă tampon. Acest lucru este util atunci cînd prima variabilă citită este de tip caracter dar nu are nici un efect în cazul în care variabila este de tip REAL sau INTEGER (deoarece caracterele EOL sînt ignorate).

5.3.3. Reprezentarea datelor în memorie

Este foarte important pentru orice programator să cunoască modul de reprezentare a fiecărui tip de date în memorie precum și necesarul de memorie în fiecare caz.

1) Intregi

Intregii ocupă fiecare cîte 2 octeți în memorie, fiind reprezentați în cod complementar.

Exemple:

```
   1 = #0001
 256 = #0100
-256 = #FF00
```

Registrul Z80 standard pentru manipularea întregilor este registrul HL.

2) Caractere, valori logice și alte valori scalare

Acestea ocupă fiecare cîte un octet, în format binar, fără semn. Pentru caractere este folosit codul ASCII extins, pe 8 biți.

Exemple:

```
'E' = #45
'[' = #5B
```

Pentru valori logice, deoarece:

```
ORD(TRUE) = 1
ORD(FALSE) = 0
```

TRUE se va reprezenta ca 1, iar FALSE prin 0. In mod analog, reprezentarea unei valori ordinale este dată de poziția ei în cadrul tipului respectiv.

3) Numere reale

Se folosește formatul cu mantisă și exponent, similar celui folosit în notația științifică standard, însă cu reprezentare binară în locul celei zecimale.

Exemple:

$2 = 2*10^0$ sau 1.0_2*2^1

$1 = 1*10^0$ sau 1.0_2*2^0

$-12.5 = -1.25*10^1$ sau $-25*2^{-1} = -11001_2*2^{-1} = -1.1001_2*2^3$

$0.1 = 1.0*10^{-1}$ sau $\frac{1}{10} = \frac{1}{1010_2} = \frac{0.1_2}{101_2} = 1.1001100_2*2^{-4}$

Un număr real este reprezentat pe 4 octeți în felul următor:

S	mantisa normalizată		exponent
23 22		0	7 0
H	*L*	*E*	*D*

unde:

S - semnul mantisei: 1 = negativ; 0 = pozitiv;
mantisa normalizată - mantisa adusă la forma 1.XXXXXX, cu bitul cel mai semnificativ (bitul 22) egal totdeauna cu 1, cu excepţia cazului cînd se reprezintă 0 (HL=0, DE=0);
exponent - exponentul este în binar, în cod complementar.

In memorie numerele reale sînt memorate sub forma EDLH.

Exemple:

```
     Semn       Mantisa               Exponent

2= 01000000  00000000  00000000  00000001
      #40       #00       #00       #01

deci         LD   HL, #4000
             LD   DE, #0100
```

4) **Inregistrări şi tablouri**

Inregistrările ocupă o cantitate de memorie egală cu totalul cerut pentru componentele sale.

In cazul *tablourilor*, dacă *n* este numărul de elemente din tablou şi s este numărul de octeţi ocupat de un element, atunci numărul de octeţi ocupat de tablou este *n**s.

Exemple:

```
ARRAY [1..10] OF INTEGER
    ocupă 10*2=20 octeţi;
ARRAY [2..12,1..10] OF CHAR
    are 11*10=110 elemente şi ocupă  110*1=110 octeţi.
```

5) **Mulţimi**

Mulţimile sînt stocate ca şiruri de biţi şi deci dacă tipul de bază are *n* elemente, atunci numărul de octeţi folosit este (*n*-1) DIV 8 +1.

Exemple:

```
SET OF CHAR necesită (256-1) DIV 8 +1=32 octeţi,
SET OF (blue,green,yellow) necesită (3-1) DIV 8 +1=1 octet.
```

6) **Pointeri - (tip referinţă)**

Pointerii ocupă 2 octeţi, care conţin adresa variabilei dinamice în format Z80, adică cu octetul cel mai puţin semnificativ în faţă.

5.3.4. Accesul direct al memoriei

HP4TM dispune de cîteva proceduri standard prin intermediul cărora este permis accesul direct la anumite zone de memorie, fie pentru a insera o rutină în cod-maşină, fie pentru a apela o rutină în cod-maşină existentă; de asemenea, execuţia programului poate fi oprită în anumite condiţii.

Procedura INLINE

Forma generală a apelului procedurii este:

INLINE(C1,C2,...,Cn);

Această procedură permite ca în programul PASCAL să fie inserată o secvență cod-mașină Z80. Valorile (C1 MOD 256, C2 MOD 256,...) sînt inserate în program direct la adresa curentă dată de contorul de locații din cadrul compilatorului. C1,C2,... sînt constante întregi sau hexa, numărul lor putînd fi oarecare.

Programul **EXINLINE** (P.V.11) exemplifică modul de folosire a procedurii INLINE. Rutina prezentată realizează transferul informației din zona [23296;8] pe ecran, și anume dacă în zona (23296,8) sînt scrise caracterele '*INLINE*' atunci rutina va scrie în linia 10, coloana 12, cu FLASH, '*INLINE*'.

Remarcăm cîteva operații în plus:

a) Prima operație este încărcarea registrului IY cu adresa 23610. In această locație, care face parte din zona variabilelor sistem, va fi pus codul de eroare în caz că există.

b) Următoarea operație o reprezintă deschiderea canalului pentru ecran. Acest lucru se realizează apelînd rutina din ROM de la adresa #1601 cu valoarea 2 în registrul A. Transmiterea caracterelor se face folosind tot o rutină din ROM de la adresa #10 cu instrucțiunea RST și cu codul caracterului în registrul A.

Linia 320 introduce o nouă funcție INCH care returnează caracterul cu codul ASCII al tastei care este apăsată în acel moment. Dacă nu este apăsată nici o tastă, funcția ia valoarea CHR(0).

```
AFAC   10 PROGRAM EXINLINE;
AFAC   20 BEGIN
AFBE   30  PAGE;
AFC3   40  INLINE(#FD,#21,#3A,#5C,#3E,2,#CD,1,#16,#3E,#16,#D7);
AFCF   50 { linia 40 realizeaza
AFCF   60          LD   IY,#5C3A ;in IY adresa cod eroare
AFCF   70          LD   A,2      ;deschidere canal
AFCF   80          CALL #1601    ;spre ecran
AFCF   90          LD   A,22     ;PRINT
AFCF  100          RST  #10      ;  AT     }
AFCF  110  INLINE(#3A,10,#D7,#3E,12,#D7,#21,0,#5B,17,#4C,#59);
AFDB  120 { linia 110 realizeaza
AFDB  130          LD   A,10     ;linia 10
AFDB  140          RST  #10
AFDB  150          LD   A,12     ;coloana 12
AFDB  160          RST  #10
AFDB  170          LD   HL,23296 ;in HL adresa de unde se transfera
AFDB  180          LD   DE,22860 ;in DE adresa zona atribute      }
AFDB  190  INLINE(6,8,#7E,#D7,#3E,#87,#12,#13,#23,5,#20,#F6,#3E,13,#D7);
AFEA  200 { linia 190 realizeaza
AFEA  210          LD   B,8
AFEA  220 CICLU    LD   A,(HL)   ;transfer caracter
AFEA  230          RST  #10
AFEA  240          LD   A,135
AFEA  250          LD   (DE),A   ;memorare atribute
AFEA  260          INC  DE       ;atribut urmator
AFEA  270          INC  HL       ;caracter urmator
AFEA  280          DEC  B        ;B<-- B-1
AFEA  290          JR   NZ,CICLU ;repeta pina cind B=0
AFEA  300          LD   A,13     ;linie noua
AFEA  310          RST  #10       }
AFEA  320  REPEAT UNTIL INCH<>CHR(0)
AFF5  330 END {$P}.
```

P.V.11.

Procedura USER

Forma generală a apelului procedurii este:

USER(V);

Este o procedură cu un singur argument *V*, întreg, care reprezintă o adresă din memorie. Procedura determină un apel (CALL) la adresa *V* din memorie. Deoarece HP4TM memorează întregii în cod complementar, pentru referiri mai mari decît 32767 (#7FFF), trebuie să folosim valori negative pentru *V*.

Exemplu:

#C000 este -16384,

deci USER (-16384) reprezintă un apel la adresa #C000. Atunci cînd ne referim la o adresă din memorie, este mai convenabil să folosim numere hexazecimale.

Rutina apelată trebuie să se încheie cu o instrucţiune Z80 RET (#C9) şi trebuie să conserve registrul IX.

Programul **EXUSER** (P.V.12) realizează exact acelaşi lucru ca şi programul **READ** (P.V.10) cu singura deosebire că el nu face parte din programul PASCAL, ci este apelat ca o rutină separată în cod-maşină. Remarcăm că RAMTOP-ul este la 59999, rutina la 60000 (#EA60). Şi în acest caz rutina trebuie să conţină operaţia a) de la procedura anterioară - în caz contrar sistemul se blochează.

```
AC35   10 PROGRAM EXUSER;
AC35   20 BEGIN
AC47   30  PAGE;
AC4C   40  USER(#EA60);
AC52   50  REPEAT UNTIL INCH<>CHR(0)
AC5D   60 END {$P}.
```

P.V.12.

Procedura POKE

Forma generală a apelului procedurii este:

POKE(X,V);

unde:

X-expresie de tip întreg reprezentînd o adresă din memorie;

V-expresie de orice tip cu excepţia tipului mulţime.

Procedura POKE stochează expresia *V* în memorie, începînd de la adresa *X*.

Exemple:

POKE(#6000,'A')	încarcă #41 în locaţia #6000;
POKE(-16384,3.6F3)	memorează succesiv valorile #00, #0B, #80, #70, în locaţiile #C000, #C001, #C002, #C003;

Programul **EXUSER** (P.V.13) exemplifică un mod de stocare în zona [23296,8] a unui set oarecare de caractere, care apoi poate fi vizualizat aşa cum s-a arătat în programul **EXINLINE** (P.V.11), folosind procedura USER(#EA60). Procedura READLN utilizată la început permite citirea într-o nouă zonă tampon, apoi sînt citite pe rînd 8 caractere şi memorate în zona indicată.

```
AC9B   10 PROGRAM EXUSER;
AC9B   20 VAR I:INTEGER;
ACA4   30     CAR:CHAR;
ACA4   40 BEGIN
ACAD   50  PAGE;
ACB2   60  READLN;
ACB5   70  FOR I:=0 TO 7 DO
ACCF   80   BEGIN
ACD2   90    READ(CAR);
ACD8  100    POKE(23296+I,CAR)
ACEB  110   END;
ACEF  120  USER(#EA60);
ACF5  130  REPEAT UNTIL INCH<>CHR(0)
AD00  140 END {$P}.
```

P.V.13.

Programul **EXPOKESIUSER** (P.V.14) realizează acelaşi lucru, dar exemplifică modul de lucru al procedurii POKE pentru un tablou. Se observă că parametrii procedurii POKE sînt doar adresa primului octet (23296) şi numele tabloului A. Memorarea întregului tablou este realizată de procedura POKE. Remarcăm că pot fi memoraţi şi întregi reprezentaţi hexa, dar în acest caz ei sînt memoraţi în format Z80, adică octetul mai puţin semnificativ la prima adresă, urmat de octetul mai semnificativ.

```
AC7B   10 PROGRAM EXPOKESIUSER;
AC7B   20 VAR A:ARRAY[1..8] OF CHAR;
AC84   30 BEGIN
AC8D   40  PAGE;
AC92   50  READLN;
AC95   60  READ(A);
AC9D   70  POKE(23296,A);
ACAA   80  USER(#EA60);
ACB0   90  REPEAT UNTIL INCH<>CHR(0)
ACBB  100 END {$P}.
```

P.V.14.

In urma rulării programului **EXPOKESIUSER** (P.V.15) pe ecran va apare scris:

'BABA/*/*'

iar memorarea s-a făcut cu:

POKE(23296,#4142)   #41='A'
POKE(23298,#4142)   #42='B'

adică:

(23296)=#42   'B'
(23297)=#41   'A'
(23298)=#42   'B'
(23299)=#41   'A'

```
AC7E   10 PROGRAM EXPOKESIUSER;
AC7E   20 BEGIN
AC90   30  PAGE;
AC95   40  POKE(23298,#4142);
ACA1   50  POKE(23300,'/*/*');
ACB5   60  POKE(23296,#4142);
ACC1   70  USER(#EA60);
ACC7   80  REPEAT UNTIL INCH<>CHR(0)
ACD2   90 END {$P}.
```

P.V.15.

Observaţie:

Pentru ca programele P.V.12 - P.V.15 să poată lucra corect este necesar ca la adresa #EA60 (60000) să existe în momentul apelului USER(#EA60) rutina în cod-maşină, dată ca şi

comentarii în programul **EXINLINE** (P.V.11), care trebuie să mai conţină la sfîrşit o instrucţiune Z80 de revenire în programul PASCAL, RET (cod #C9).

Procedura OUT

Forma generală a apelului procedurii este:

OUT(P,C);

unde:

p-este un parametru de tip întreg, valoarea lui este încărcată în registrul BC;

c-este un parametru de tip caracter; valoarea lui este încărcată în registrul A, după care se execută instrucţiunea Z80: OUT A, (C).

Această procedură este folosită pentru a avea acces direct la porturile de ieşire ale microprocesorului Z80, fără a fi nevoie să folosim procedura INLINE.

Exemplu:

OUT(1,'A')-transmite caracterul 'A' la portul 1 al microprocesorului Z80.

In programul **EXOUT** (P.V.16) se pot urmări diferite efecte sonore şi efecte pe *border*-ul ecranului, mărind sau micşorînd ciclul care simulează instrucţiunea PAUSE, sau schimbînd valorile transmise la portul 254.

```
ACAE   PROGRAM EXOUT;
ACAE   VAR I,J:INTEGER;
ACB7
ACB7   BEGIN
ACC0    FOR I:= 1 TO 3200 DO
ACDA     BEGIN
ACDD      OUT(254,CHR(123));
ACE8      FOR J:=1 TO 25 DO J:=J+1;
AD0F      OUT(254,CHR(97));
AD1A      FOR J:=1 TO 25 DO J:=J+1;
AD41      OUT(254,CHR(255))
AD4A     END;
AD4F   END {$P}.
```

P.V.16.

Procedura HALT

Forma generală a apelului procedurii este:

HALT;

In urma execuţiei procedurii programul se va opri cu mesajul:

'HALT at PC=XXXX'

unde XXXX este adresa hexazecimală a locaţiei de memorie de unde a fost emis HALT. Instrucţiunea HALT se foloseşte în depanarea programelor. Punînd instrucţiunea HALT pe fiecare din ramurile unui program se va putea determina ramura pe care se execută programul.

5.3.5. Proceduri HP4TM pentru lucrul cu caseta

Există două proceduri standard specifice implementării HP4TM pentru memorarea, respectiv încărcarea datelor de pe caseta magnetică.

Procedura TOUT

Forma generală a apelului procedurii este:

TOUT(NUME,ADRESA,LUNGIME);

Procedura TOUT (Tape OUTput) este folosită pentru a salva pe bandă variabile, zone de memorie etc. Primul parametru (*NUME*) este de tip ARRAY[1..8] OF CHAR și reprezintă denumirea fișierului care urmează să fie salvat. Vor fi salvați pe casetă *LUNGIME* octeți, începînd de la adresa *ADRESA*. Ambii parametri (*ADRESA* și *LUNGIME*) trebuie să fie de tip întreg.

Exemplu:
pentru a salva un SCREEN$ se poate folosi:
TOUT('ECRAN   ',16384,6912);

Bineînțeles, în același mod pot fi salvate variabile sau tablouri, cu condiția să fie cunoscută adresa și lungimea variabilei respective. Deoarece în PASCAL HP4TM există două funcții standard care permit aflarea adresei și lungimii unei variabile (vor fi studiate în paragraful 5.4), se poate folosi procedura TOUT și în felul următor:
TOUT('VAR V   ',ADDR(V),SIZE(V));
care realizează salvarea pe casetă a variabilei *V* sub numele 'VAR V   ', de la adresa ADDR(V) și avînd lungimea SIZE(V).

Programul **EXTOUTTIN** (P.V.17) constituie un exemplu de utilizare a procedurilor PAGE, WRITE, WRITELN, POKE pentru realizarea unui tabel pe ecran care conține valorile întregi I, ale pătratului variabilei, precum și ale radicalului variabilei cu 6 zecimale exacte. Pentru calculul radicalului s-a folosit funcția standard aritmetică SQRT.

```
ADC1   10 PROGRAM EXTOUTTIN;
ADC1   20 VAR I:INTEGER;
ADCA   30 BEGIN
ADD3   40  PAGE;
ADD8   50  WRITE(CHR(22),CHR(1);CHR(3));
ADED   60  WRITELN(' I         I*I          I^(1/2)');
AE16   70  WRITE(CHR(22),CHR(4),CHR(1));
AE2B   80  WRITELN;
AE2E   90  FOR I:=1 TO 17 DO
AE48  100   WRITELN(I:5,I*I:10,SQRT(I):15:6);
AE87  110  I:=16384;
AE8D  120  REPEAT
AE8D  130   POKE(I,128);
AE9C  140   POKE(I+31,1);
AEB1  150   I:=I+32;
AEC0  160  UNTIL I>22495;
AED3  170  I:=0;
AED9  180  REPEAT
AED9  190   POKE(16384+I,#FFFF);
AEF3  200   POKE(16480+I,#FFFF);
AF0A  210   POKE(22496+I,#FFFF);
AF21  220   I:=I+2;
AF29  230  UNTIL I>30;
AF3C  240  REPEAT UNTIL INCH <>CHR(0);
AF51  250  TOUT('ECRAN   ',16384,6912)
AF6E  260 END {$P}.
```

```
AC42   10 PROGRAM EXTOUTTIN;
AC42   20 BEGIN
AC54   30  PAGE;
AC59   40  TIN('ECRAN   ',16384);
AC6F   50  REPEAT UNTIL INCH<>CHR(0)
AC7A   60 END {$P}.
```

I	I*I	I↑(1/2)
1	1	1.000000
2	4	1.414213
3	9	1.732051
4	16	2.000000
5	25	2.236068
6	36	2.449489
7	49	2.645751
8	64	2.828427
9	81	3.000000
10	100	3.162277
11	121	3.316624
12	144	3.464102
13	169	3.605551
14	196	3.741657
15	225	3.872983
16	256	4.000000
17	289	4.123106

P.V.17.

Procedura TIN

Forma generală a apelului procedurii este:

TIN(NUME,ADRESA);

Procedura TIN (Tape INput) este folosită pentru a încărca de pe bandă variabile sau zone de memorie care au fost salvate cu TOUT. Parametrul *NUME* este de tip ARRAY[1..8] OF CHAR, iar *ADRESA* este de tip întreg. Se caută pe bandă fişierul cu denumirea *NUME*, care este apoi încărcat în memorie de la adresa *ADRESA*. Numărul de octeţi care se încarcă este luat din *header*-ul fişierului (salvat pe casetă cu TOUT).

Exemplu:

pentru a încărca variabila *V* salvată anterior cu TOUT, se foloseşte:

TIN('VAR V ',ADDR(V));

Observaţie:

Din cauză că fişierele sursă sînt înregistrate de editor în acelaşi format cu cel folosit de procedurile TOUT şi TIN, cu TIN pot fi încărcate fişierele de text în variabile de tip tablou de caractere pentru prelucrare ulterioară.

5.3.6. Alocarea dinamică a memoriei

Alocarea dinamică a memoriei va fi tratată în detaliu în capitolul VI unde vor fi prezentate în amănunt şi procedurile aferente: NEW, MARK, RELEASE.

5.3.7. Proceduri definite în program

Utilizarea procedurilor standard oferă posibilităţi largi programatorilor în rezolvarea de probleme, dar puterea limbajului PASCAL constă tocmai în posibilităţile nelimitate de concepere a procedurilor *proprii* fiecărui utilizator. Aceste proceduri se definesc în program, iar *după* ce au fost definite conform diagramelor din paragraful 5.3.1 pot fi folosite ca şi procedurile standard obişnuite.

Programul **MEDIEINMATRICE** (P.V.18) calculează media elementelor maxime de pe fiecare linie a unei matrici. Programul conține două proceduri definite de utilizator:

ELMAX - liniile 130 - 190
MEDIE - liniile 210 - 280

Prima procedură, ELMAX, are trei parametri formali: primul parametru, I - este transmis prin valoare; el are rolul de a indica linia în care se calculează maximul; ceilalți doi parametri sînt A (reprezentînd matricea în care se lucrează), respectiv VALMAX (care va transmite la ieșire elementul maxim din linia respectivă). Se observă că ultimii doi parametri sînt precedați de VAR, deci sînt parametri transmiși prin referință, lucru obligatoriu pentru VALMAX, pentru a putea transmite la ieșire valoarea calculată.

A doua procedură, MEDIE, are doi parametri formali, ambii transmiși prin referință - vectorul V, (care conține maximele de pe fiecare linie), respectiv VALMED, (care va conține la ieșire valoarea medie a maximelor). Transmiterea parametrului VALMED prin referință este obligatorie, acesta fiind un parametru de ieșire.

Este de remarcat, de asemenea, faptul că în afara parametrilor formali, în descrierea unei proceduri pot să apară și alte nume, reprezentînd diferite variabile. In programul **MEDIEINMATRICE** (P.V.18), procedura MEDIE folosește variabilele locale I și X și constanta globală N, definite în secțiunea de declarații a blocului program. In același program identificatorul I se utilizează atît ca variabilă globală în blocul programului principal, cît și ca variabilă locală în procedura MEDIE. Remarcăm că în momentul declarării variabilei locale I, variabila globală cu același nume din blocul apelant se conservă și orice modificare a valorii variabilei locale I afectează doar variabila din procedură.

```
B025  10 { PROGRAMUL CALCULEAZA MEDIA ELEMENTELOR MAXIME
B025  20   DE PE FIECARE LINIE A UNEI MATRICI          }
B025  30
B025  40 PROGRAM MEDIEINMATRICE;
B025  50 CONST N=5;
B025  60 TYPE LINIE=ARRAY[1..N] OF REAL;
B025  70      MATRICE=ARRAY[1..N,1..N] OF REAL;
B025  80 VAR X:REAL;
B02E  90     VECTOR:LINIE;
B02E 100     MAT:MATRICE;
B02E 110     I,J:1..N;
B02E 120
B02E 130 PROCEDURE ELMAX(I:INTEGER;VAR A:MATRICE;VAR VALMAX:REAL);
B031 140 VAR J:1..N;
B031 150 BEGIN
B049 160  VALMAX:=A[I,1];
B09F 170  FOR J:=2 TO N DO
B0C2 180   IF VALMAX<A[I,J] THEN VALMAX:=A[I,J];
B188 190 END;
B193 200
B193 210 PROCEDURE MEDIE(VAR V:LINIE;VAR VALMED:REAL);
B196 220 VAR I:1..N;
B196 230     X:REAL;
B196 240 BEGIN
B1AE 250  X:=0;
B1C0 260  FOR I:=1 TO N DO X:=X+V[I];
B22F 270  VALMED:=X/N
B24C 280 END;
B266 290
B266 300 BEGIN
B26F 310  FOR I:=1 TO N DO
B289 320   BEGIN
B28C 330    FOR J:=1 TO N DO
```

```
B2A6        BEGIN
B2A9         READ(X);
B2B3         MAT[I,J]:=X
B2EB        END;
B2FF       WRITELN;
B302
B302  { *** apel procedura ELMAX *** }
B302
B302       ELMAX(I,MAT,X);
B313       VECTOR[I]:=X
B32F     END;
B343    PAGE;
B348    WRITE(CHR(16));
B34F    FOR I:=1 TO N DO
B369     BEGIN
B36C      FOR J:=1 TO N DO
B386       WRITE(MAT[I,J]:6:2);
B3D8      WRITELN
B3D8     END;
B3DE
B3DE  { *** apel procedura MEDIE *** }
B3DE
B3DE    MEDIE(VECTOR,X);
B3EB    WRITELN;
B3EE    WRITELN('ELEMENTELE MAXIME ALE LINIILOR');
B41A    FOR J:=1 TO N DO WRITE(VECTOR[J]:6:2);
B46A    WRITELN;WRITELN;
B470    WRITELN('MEDIA ELEMENTELOR MAXIME: ',X:6:2);
B4AC    WRITELN;WRITE(CHR(16))
B4B6  END {$P}.
```

```
  4.00  5.00  7.00 -8.00  0.00
  2.00 -7.00 11.00 23.00  9.00
 -9.00  5.00 23.00 -1.00  5.00
 22.00 -8.00  0.00 -7.00  1.00
 -6.00 34.00 -7.00  0.00  3.00

ELEMENTELE MAXIME ALE LINIILOR
  7.00 23.00 23.00 22.00 34.00

MEDIA ELEMENTELOR MAXIME:  21.80
```

P.V.18.

Observaţii:

1) O procedură poate avea un număr oarecare de parametri, eventual nici unul - de exemplu dacă în procedură doar se afişează mesaje. De asemenea vom avea proceduri definite fără parametri în acele cazuri cînd variabilele tratate în procedură sînt globale.

Exemplu:

```
PROGRAM PP;
VAR A,B:REAL;
.............
PROCEDURE P;
............
END;{P}
BEGIN {PP}
............
  P;
............
END.
```

In procedura P se pot trata variabilele A şi B ale căror valori noi se regăsesc în blocul apelant după fiecare apel al procedurii P. Remarcăm însă că de obicei procedurile vor avea parametri, ele prelucrînd diferite valori corespunzător fiecărui apel. Astfel dacă procedura P va prelucra, corespunzător unui apel variabila A, iar pentru alt apel variabila B, structura programului va fi:

```
PROGRAM PP;
VAR A,B:REAL;
.............
PROCEDURE P (VAR X:REAL);
.............
END;{P}
BEGIN {PP}
.............
  P(A);
.............
  P(B)
.............
END.
```

In acest exemplu, parametrul formal X ar putea fi orice identificator, chiar şi A sau B, acesta fiind folosit doar pentru descrierea operaţiilor asupra valorilor transmise din blocul apelant.

2) In afara declaraţiei variabilelor, într-o procedură pot să apară şi alte definiţii şi declaraţii (de etichete, constante, tipuri, proceduri şi funcţii). Deci structura declaraţiei unei proceduri este similară cu cea a programului (procedurii principale), descrisă în capitolul I. Deosebirea constă în faptul că etichetele, constantele, tipurile, variabilele, procedurile şi funcţiile definite sau declarate într-o procedură sînt locale procedurii respective. Domeniul de valabilitate a fiecărei entităţi se stabileşte conform regulilor prezentate în paragraful 5.1.

Exemplu:

```
 10 PROGRAM PP;
 20 TYPE T=-5..5;
 30 VAR A:INTEGER;
 40     B:T;
 50
 60 PROCEDURE P1(T:REAL);
 70 LABEL 11;
 80
 90 PROCEDURE P2;
100 VAR B:T;
110 BEGIN {P2}
120   A:=B
130 END; {P2}
140
150 BEGIN {P1}
160   B:=T*A
170 END; {P1}
180
190 BEGIN {PP}
200   READ(A,B);
210     IF A<B THEN GOTO 11;
220     P2;
230 11: P1(A/B)
240 END. {PP}
```

Observaţii:

1) linia 100 va produce eroare - tipul T nu este tip pentru P1, deoarece identificatorul T este redeclarat în procedura P1 ca o variabilă reală;

2) în linia 120 A este din PP, iar B este din P2 dar este

nedefinită;

3) în linia 160 A și B sînt din PP, iar T este din P1 (parametru formal);

4) linia 210 va produce eroare - eticheta 11 este definită în P1, ea nu există în PP, este nedefinită;

5) linia 220 va produce eroare - procedura P2 este nedefinită în afara procedurii P1;

6) în linia 230 procedura P1, variabilele A și B sînt din PP, iar eticheta 11 este nedefinită.

Directiva FORWARD

Să considerăm un program în care două proceduri se apelează reciproc:

```
10 PROGRAM APELRECIPROC;
   {declarații}
100 PROCEDURE P(X:REAL);
110  {declarații P}
120 BEGIN
     ...
150 Q(X);
     ...
190 END;  {sfîrșit P}
200 PROCEDURE Q(Y:REAL);
210 {declarații Q}
220 BEGIN
     ...
250 P(Y);
     ...
290 END;{sfîrșit Q}
300 BEGIN
     ...
900 END.
```

Un program cu structura de mai sus nu este acceptat de compilatorul PASCAL care cere ca la apelul unei proceduri aceasta să fie declarată cu întregul său bloc. Pentru a putea rezolva această situație, care apare relativ frecvent în practica programării în PASCAL, se utilizează directiva FORWARD, așa cum se poate vedea în diagrama de sintaxă din paragraful 5.3.1. Rolul acestei directive este de a declara un identificator ca nume de procedură impreună cu antetul său permițînd astfel apelul său inainte ca să fie dezvoltată întregul bloc al său.

In concluzie, structura variantei corecte a programului din exemplul anterior va fi următoarea:

```
 10 PROGRAM APELRECIPROC;
      {declaratii}
 90 PROCEDURE Q(Y:REAL);FORWARD;
100 PROCEDURE P(X:REAL);
110   {declaratii}
120 BEGIN
     ...
150 Q(X);
     ...
190 END;  {sfirsit P}
200 PROCEDURE Q;
210 {declaratii Q}
220 BEGIN
```

```
...
250 P(Y);
...
290 END;  {sfirsit Q}
300 BEGIN
...
900 END.
```

Se observă că procedura apelată înainte ca blocul său să fie dezvoltat este doar declarată prin antetul său, urmat de directiva FORWARD (linia 90). Pe de altă parte, declaraţia efectivă a procedurii, care a fost anterior "semnalată" cu FORWARD, nu conţine lista parametrilor formali. In blocul procedurii se vor folosi parametrii formali din declaraţia de procedură cu FORWARD.

Programul **SPATII** (P.V.39) oferă un exemplu care necesită folosirea declaraţiei anticipate a unei proceduri.

In finalul acestui paragraf prezentăm un program mai complex care calculează şi afişează în ordine alfabetică şi în ordinea descrescătoare a mediilor situaţia trimestrială a unei clase în care sînt cel mult 36 elevi, numărul de materii fiind maximum 16.

In programul **MEDIEINCLASA** (P.V.19) sînt utilizate 5 proceduri, fiecare îndeplinind o funcţie indicată sugestiv prin numele procedurii:

CITNUM	- liniile	140	-	210
CITNOTE	- liniile	230	-	360
ORDONNUME	- liniile	380	-	570
ORDONNOTA	- liniile	590	-	780
LISTARE	- liniile	800	-	850

```
B1D4   20 PROGRAM MEDIEINCLASA;
B1D4   30 TYPE NUME=ARRAY[1..16] OF CHAR;
B1D4   40      MEDII=ARRAY[1..16] OF INTEGER;
B1D4   50      ELEV=RECORD
B1D4   60        NUM:NUME;
B1D4   70        MEDIA:REAL
B1D4   80      END;
B1D4   90      CLASA=ARRAY[1..36] OF ELEV;
B1D4  100 VAR CL:CLASA;
B1DD  110     N:INTEGER;
B1DD  120     I:1..36;
B1DD  130
B1DD  140 PROCEDURE CITNUM(VAR SIR:NUME);
B1E0  150 VAR I:INTEGER;
B1E0  160 BEGIN
B1F8  170  READLN;
B1FB  180  READ(SIR);
B206  190  FOR I:=1 TO 16 DO
B229  200   IF SIR[I]=CHR(0) THEN SIR[I]:=' ';
B281  210 END;
B289  220
B289  230 PROCEDURE CITNOTE(VAR MEDIE:REAL);
B28C  240 VAR S,I:INTEGER;
B28C  250     NOTE:MEDII;
B28C  260 BEGIN
B2A4  270  S:=0;
B2AD  280  FOR I:=1 TO 16 DO
B2D0  290   BEGIN
B2D3  300    WRITE('MARK NO.',I:2,':');
B2FA  310    READ(NOTE[I]);
B324  320    S:=S+NOTE[I];
B35D  330   END;
B360  340  IF NOTE[11]=0 THEN MEDIE:=S/15
B3AB  350     ELSE MEDIE:=S/16
B3D3  360 END;
B3EA  370
```

```
B3EA PROCEDURE ORDONNUME(VAR C:CLASA);
B3ED VAR I:1..35;
B3ED     SW:BOOLEAN;
B3ED     AUX:ELEV;
B3ED BEGIN
B405 REPEAT
B405  SW:=FALSE;
B40C  I:=1;
B415  REPEAT
B415   IF C[I].NUM>C[I+1].NUM THEN
B47A      BEGIN
B47A       AUX:=C[I];
B4AD       C[I]:=C[I+1];
B4FF       C[I+1]:=AUX;
B533       SW:=TRUE;
B538      END;
B538   I:=I+1;
B545  UNTIL I>N-1;
B55C UNTIL NOT SW;
B565 END;
B571
B571 PROCEDURE ORDONNOTA(VAR C:CLASA);
B574 VAR I:1..35;
B574     SW:BOOLEAN;
B574     AUX:ELEV;
B574 BEGIN
B58C REPEAT
B58C  SW:=FALSE;
B593  I:=1;
B59C  REPEAT
B59C   IF C[I].MEDIA<C[I+1].MEDIA THEN
B613      BEGIN
B613       AUX:=C[I];
B646       C[I]:=C[I+1];
B698       C[I+1]:=AUX;
B6CC       SW:=TRUE;
B6D1      END;
B6D1   I:=I+1;
B6DE  UNTIL I>N-1;
B6F5 UNTIL NOT SW;
B6FE END;
B70A
B70A PROCEDURE LISTARE(C:CLASA);
B70D VAR I:INTEGER;
B70D BEGIN
B725  FOR I:=1 TO N DO
B74C    WRITELN(I:2,'. ',C[I].NUM,' - ',C[I].MEDIA:5:2)
B7E2 END;
B7F4
B7F4 BEGIN
B7FD  READ(N);
B803  FOR I:=1 TO N DO
B821   BEGIN
B824    WRITELN('NAME PLEASE:');
B83E    CITNUM(CL[I].NUM);
B863    WRITELN('MARK PLEASE:');
B87D    CITNOTE(CL[I].MEDIA)
B8A1   END;
B8AA  ORDONNUME(CL);
B8B3  PAGE;
B8B8  WRITELN(' *** ORDONARE ALFABETICA ***');WRITELN;
B8E5  LISTARE(CL);
B8FB  ORDONNOTA(CL);
B904  WRITELN;WRITELN(' *** ORDONARE DUPA MEDIE ***');WRITELN;
B934  LISTARE(CL)
B945 END {$P}.
```

5.4. FUNCTII

5.4.1. Declararea funcţiilor

Funcţiile sînt subprograme care *calculează şi returnează* subprogramului apelant *o singură valoare*. Pentru limbajul PASCAL, această valoare este în mod obligatoriu de *tip simplu* sau *referinţă* (capitolul VI). Această valoare se transmite în locul apelului funcţiei care astfel apare ca un operand într-o expresie oarecare.

O declaraţie de funcţie poate fi identificată într-un program conform diagramei de sintaxă din fig.5.7.

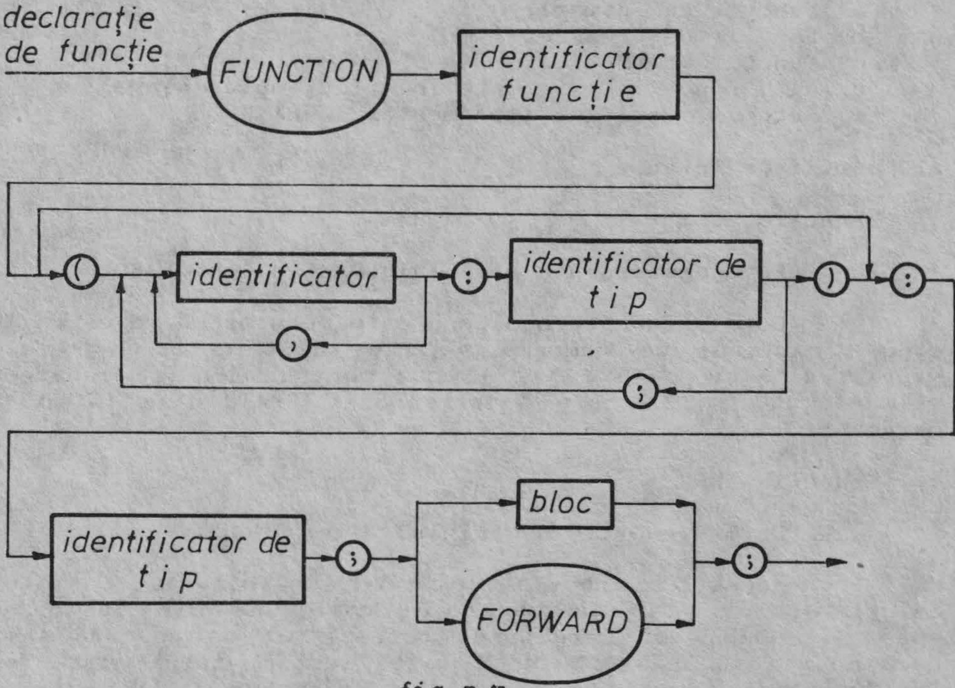


fig.5.7

Din diagramă se observă că există două forme ale declaraţiei de funcţie:

-funcţie fără parametri;

-funcţie cu parametri.

In cazul funcţiei cu parametri, lista parametrilor formali este identică cu cea de la proceduri.

Sintaxa declaraţiei funcţiei seamănă cu sintaxa declaraţiei procedurii. Rezultă deci că observaţiile privind parametrii formali şi efectivi sau domeniul de valabilitate este aplicabil şi în cazul funcţiei.

Rezultatul calculat de o funcţie este asociat numelui funcţiei, căruia conform diagramei de sintaxă, îi este precizat tipul. Numele funcţiei se va referi *în blocul funcţiei* ca orice variabilă. Nu se vor specifica parametrii decît în caz de apel recursiv. Din această cauză *este obligatoriu ca numele funcţiei să apară cel puţin o dată în partea stîngă a unei instrucţiuni de atribuire de forma*:

identificator-funcţie:=expresie;

Ultima valoare atribuită lui *identificator-funcţie* va fi rezultatul funcţiei.

In momentul apelului unei funcţii, numele funcţiei apare ca *operand într-o expresie*, urmat de parametrii actuali.

5.4.2. Funcţii standard specifice limbajului PASCAL HP4TM

Asemenea procedurilor standard, în limbajul PASCAL există o serie de funcţii standard, (care evident nu trebuie declarate), ele asigurînd o mulţime de facilităţi necesare oricărui program. Aceste funcţii se pot grupa astfel:

a) funcţii de intrare;
b) funcţii de transfer;
c) funcţii aritmetice;
d) alte funcţii (existente în orice implementare);
e) funcţii specifice implementării HP4TM.

a) Funcţii de intrare

Funcţia EOLN

Forma generală a apelului funcţiei este: EOLN.

Este o funcţie booleană care returnează valoarea TRUE cînd caracterul ce urmează să fie citit este un caracter *sfîrşit-de-linie* (CHR(13)). In caz contrar funcţia returnează valoarea FALSE. In cazul impementării HP4TM funcţia nu are parametri.

Funcţia INCH

Forma generală a apelului este: INCH.

Această funcţie execută baleierea tastaturii calculatorului, iar în cazul în care una din taste este apăsată, returnează caracterul reprezentat de această tastă. Dacă nu a fost apăsată nici o tastă, este returnat caracterul CHR(0). Deci funcţia este de tip caracter (CHAR). Nu are parametri.

Programul **TESTEOLNINCH** (P.V.20a) exemplifică un mod de utilizare a funcţiilor EOLN şi INCH. Execuţia se opreşte cînd este apăsată o anumită tastă (în acest caz tasta C).

Se observă că introducerea şirului de caractere 'EOLN' urmat de <CR> va produce scrierea următorului rezultat:

FALSE E FALSE O FALSE L FALSE N TRUE

```
ACCC  PROGRAM TESTEOLNINCH;
ACCC  CONST C=CHR(13);
ACCC        P=CHR(16);
ACCC  VAR CAR:CHAR;
ACD5  BEGIN
ACDE   REPEAT
ACDE    READLN;
ACE4    REPEAT
ACE4     WRITE(P,EOLN:6,P);
AD04     READ(CAR);
AD0A     WRITE(P,CAR:2,P);
AD24    UNTIL EOLN;
AD2B    WRITE(P,EOLN:6,C,P);
AD4D    REPEAT UNTIL INCH IN ['C'];
AD77   UNTIL FALSE
AD79  END {$P}.
```

```
FALSE E FALSE O FALSE L FALSE N  TRUE
FALSE T FALSE E FALSE S FALSE T FALSE A FALSE R FALSE E  TRUE
```

P.V.20a

Dacă eliminăm linia 70, care are rol de a citi într-o nouă zonă tampon (vezi procedura READ, paragraful 5.3.2), atunci înainte de a introduce şirul 'EOLN' va apare scris TRUE şi programul aşteaptă introducerea şirului, deoarece zona tampon iniţială are primul caracter CHR(13), deci acesta va fi primul citit (**TESTEOLNINCH** - P.V.20b).

```
ACD0   10 PROGRAM TESTEOLNINCH;
ACD0   20 CONST C=CHR(13);
ACD0   30       P=CHR(16);
ACD0   40 VAR CAR:CHAR;
ACD9   50 BEGIN
ACE2   60  REPEAT
ACE2   70 { READLN; }
ACE2   80   REPEAT
ACE5   90    WRITE(P,EOLN:6,P);
AD05  100    READ(CAR);
AD0B  110    WRITE(P,CAR:2,P);
AD25  120   UNTIL EOLN;
AD2C  130   WRITE(P,EOLN:6,C,P);
AD4E  140   REPEAT UNTIL INCH IN ['C'];
AD78  150  UNTIL FALSE
AD7A  160 END {$P}.
```

```
  TRUE
 FALSE E FALSE O FALSE L FALSE N  TRUE
  TRUE
 FALSE T FALSE E FALSE S FALSE T FALSE A FALSE R FALSE E  TRUE
  TRUE
```

P.V.20b

b) Funcţii de transfer

Funcţia TRUNC

Forma generală a apelului funcţiei este: TRUNC(X).

Funcţia are un parametru de tip real sau întreg, returnează cel mai mare întreg, mai mic sau egal cu X, pentru X pozitiv, sau cel mai mic întreg, mai mare sau egal cu X, dacă X este negativ. Deci funcţia este de tip întreg.

Funcţia ROUND

Forma generală a apelului este: ROUND(X).

Funcţia are un parametru de tip real sau întreg. Ea returnează un rezultat de tip întreg şi anume cel mai apropiat întreg faţă de X, în conformitate cu regulile standard de rotunjire.

Funcţia ENTIER

Forma generală a apelului funcţiei este: ENTIER(X).

Funcţia are un parametru de tip real sau întreg şi returnează un rezultat de tip întreg - partea întreagă a lui X.
Observaţie:
Funcţia ENTIER este specifică implementării HP4TM. Este echivalentă cu INT din BASIC şi este utilă cînd se scriu rutine rapide, pentru aplicaţii în matematică.

Funcţia ORD

Forma generală a apelului funcţiei este: ORD(X).

Are un parametru de tip ordinal. Funcţia returnează un rezultat de tip întreg care reprezintă numărul de ordine a valorii lui X în cadrul setului care defineşte tipul lui X. Dacă X este de tip întreg, atunci ORD(X)=X.

Funcţia CHR

Forma generală a apelului funcţiei este: CHR(X).

Funcţia are un parametru de tip întreg şi returnează caracterul al cărui cod ASCII este X.

Din definiţia funcţiilor de transfer ORD şi CHR se observă că există relaţiile:

ORD(CHR(i))=i şi

CHR(ORD(c))=c

Observaţie:

In cazul funcţiilor cu parametru (prezentate anterior), parametrul actual poate fi o *expresie* de tipul cerut de funcţie.

Programul **FUNCTIITRANSFER** (P.V.21) ilustrează modul de lucru cu funcţiile prezentate.

```
AF68   10 PROGRAM FUNCTIITRANSFER;
AF68   20 TYPE TRANSFER=(TRUN,ROUN,ENTIE,ORDIN,CAHR);
AF68   30 VAR X,PAS:REAL;
AF71   40     I:INTEGER;
AF71   50     F:TRANSFER;
AF71   60
AF71   70 PROCEDURE TRUNCROUNDENTIER;
AF74   80 BEGIN
AF8C   90  X:=-2.1;
AFA1  100  PAS:=0.1;
AFAE  110  WHILE X<6 DO
AFCF  120  BEGIN
AFCF  130   WRITE('  TRUNC(',X:4:1,')=',TRUNC(X):2);
B016  140   WRITE('  ROUND(',X:4:1,')=',ROUND(X):2);
B05D  150   WRITELN('  ENTIER(',X:4:1,')=',ENTIER(X):2);
B0A8  160   PAS:=PAS+0.1;
B0C1  170   X:=X+PAS
B0CA  180  END
B0DB  190 END;
B0E4  200
B0E4  210 PROCEDURE FUNCTIAORD;
B0E7  220 BEGIN
B0FF  230  WRITELN('  ORD(FALSE)=',ORD(FALSE):1);
B127  240  WRITELN('  ORD(TRUE)=',ORD(TRUE):1);
B14F  250  WRITELN('  ORD(ORDIN)=',ORD(ORDIN):1);
B178  260  I:=-1;
B181  270  WRITELN('  ORD(',I:2,')=',ORD(I):1)
B1B7  280 END;
B1C0  290
B1C0  300 PROCEDURE FUNCTIACHR;
B1C3  310 BEGIN
B1DB  320  FOR I:=32 TO 63 DO
B1F5  330   BEGIN
B1F8  340    WRITE('CHR(',I:3,')=',CHR(I):1);
B231  350    WRITE(' CHR(',I+32:3,')=',CHR(I+32):1);
B27D  360    WRITELN(' CHR(',I+64:3,')=',CHR(I+64):1)
B2C9  370   END
B2CC  380 END;
B2D5  390
B2D5  400 BEGIN
B2DE  410  WRITELN(CHR(16));WRITELN;
B2EB  420  TRUNCROUNDENTIER;WRITELN;
B2F3  430  FUNCTIAORD;WRITELN;
B2FB  440  FUNCTIACHR;WRITELN;
B303  450  WRITELN(CHR(16))
B30A  460 END {$P}.
```

```
TRUNC(-2.1)=-2  ROUND(-2.1)=-2  ENTIER(-2.1)=-3
TRUNC(-1.9)=-1  ROUND(-1.9)=-2  ENTIER(-1.9)=-2
TRUNC(-1.6)=-1  ROUND(-1.6)=-2  ENTIER(-1.6)=-2
TRUNC(-1.2)=-1  ROUND(-1.2)=-1  ENTIER(-1.2)=-2
TRUNC(-0.7)= 0  ROUND(-0.7)=-1  ENTIER(-0.7)=-1
TRUNC(-0.1)= 0  ROUND(-0.1)= 0  ENTIER(-0.1)=-1
TRUNC( 0.6)= 0  ROUND( 0.6)= 1  ENTIER( 0.6)= 0
TRUNC( 1.4)= 1  ROUND( 1.4)= 1  ENTIER( 1.4)= 1
TRUNC( 2.3)= 2  ROUND( 2.3)= 2  ENTIER( 2.3)= 2
TRUNC( 3.3)= 3  ROUND( 3.3)= 3  ENTIER( 3.3)= 3
TRUNC( 4.4)= 4  ROUND( 4.4)= 4  ENTIER( 4.4)= 4
TRUNC( 5.6)= 5  ROUND( 5.6)= 6  ENTIER( 5.6)= 5

ORD(FALSE)=0
ORD(TRUE)=1
ORD(ORDIN)=3
ORD(-1)=-1

CHR( 32)=  CHR( 64)=@ CHR( 96)=`
CHR( 33)=! CHR( 65)=A CHR( 97)=a
CHR( 34)=" CHR( 66)=B CHR( 98)=b
CHR( 35)=# CHR( 67)=C CHR( 99)=c
CHR( 36)=$ CHR( 68)=D CHR(100)=d
CHR( 37)=% CHR( 69)=E CHR(101)=e
CHR( 38)=& CHR( 70)=F CHR(102)=f
CHR( 39)=' CHR( 71)=G CHR(103)=g
CHR( 40)=( CHR( 72)=H CHR(104)=h
CHR( 41)=) CHR( 73)=I CHR(105)=i
CHR( 42)=* CHR( 74)=J CHR(106)=j
CHR( 43)=+ CHR( 75)=K CHR(107)=k
CHR( 44)=, CHR( 76)=L CHR(108)=l
CHR( 45)=- CHR( 77)=M CHR(109)=m
CHR( 46)=. CHR( 78)=N CHR(110)=n
CHR( 47)=/ CHR( 79)=O CHR(111)=o
CHR( 48)=0 CHR( 80)=P CHR(112)=p
CHR( 49)=1 CHR( 81)=Q CHR(113)=q
CHR( 50)=2 CHR( 82)=R CHR(114)=r
CHR( 51)=3 CHR( 83)=S CHR(115)=s
CHR( 52)=4 CHR( 84)=T CHR(116)=t
CHR( 53)=5 CHR( 85)=U CHR(117)=u
CHR( 54)=6 CHR( 86)=V CHR(118)=v
CHR( 55)=7 CHR( 87)=W CHR(119)=w
CHR( 56)=8 CHR( 88)=X CHR(120)=x
CHR( 57)=9 CHR( 89)=Y CHR(121)=y
CHR( 58)=: CHR( 90)=Z CHR(122)=z
CHR( 59)=; CHR( 91)=[ CHR(123)={
CHR( 60)=< CHR( 92)=\ CHR(124)=|
CHR( 61)== CHR( 93)=] CHR(125)=}
CHR( 62)=> CHR( 94)=^ CHR(126)=~
CHR( 63)=? CHR( 95)=_ CHR(127)
```

P.V.21.

c) Funcţii aritmetice

Toate funcţiile aritmetice au un singur parametru, care trebuie să fie de tip real sau întreg.

Funcţia ABS

Forma generală a apelului funcţiei este: ABS(X).

Funcţia returnează un rezultat de acelaşi tip cu X, şi anume valoarea absolută a lui X.

Funcţia SQR

Forma generală a apelului este: SQR(X).

Funcţia returnează pătratul lui X, avînd acelaşi tip ca variabila X.

Funcţia SQRT

Forma generală a apelului funcţiei este: SQRT(X).

Funcţia returnează rădăcina pătrată a lui X, rezultatul fiind întotdeauna de tip real. In cazul în care parametrul X este negativ, se generează mesajul de eroare 'Math call error'.

Funcţia FRAC

Forma generală a apelului funcţiei este: FRAC(X).

Funcţia returnează o valoare de tip real, egală cu partea fracţionară a lui X, deci avem relaţia:
FRAC(X)=X-ENTIER(X).

Funcţiile trigonometrice: SIN, COS şi TAN

Forma generală a apelurilor este: SIN(X), COS(X), TAN(X).

Funcţiile returnează valorile funcţiilor trigonometrice corespunzătoare. Parametrul X se dă în radiani. Rezultatul funcţiilor este întotdeauna de tip real.

Funcţia ARCTAN

Forma generală a apelului funcţiei este: ARCTAN(X).

Funcţia, de tip real returnează, în radiani, unghiul a cărui tangentă este egală cu X.

Funcţia EXP

Forma generală a apelului funcţiei este: EXP(X).

Funcţia returnează valoarea lui e^x (e este baza logaritmului natural). Rezultatul este întotdeauna de tip real.

Funcţia LN

Forma generală a apelului funcţiei este: LN(X).

Funcţia returnează logaritmul natural (în baza e) al lui X. Rezultatul este întotdeauna de tip real. Dacă X<=0, se generează mesajul 'Math call error'.

Programul **FUNCTIIARITMETICE** (P.V.22) ilustrează modul de lucru cu funcţiile aritmetice.

```
AFBB   10 PROGRAM FUNCTIIARITMETICE:
AFBB   20 VAR X,PAS:REAL;
AFC4   30     I:INTEGER:
AFC4   40
AFC4   50 PROCEDURE ARITMETICE:
AFC7   60 BEGIN
AFDF   70  X:=-2.1:
AFF4   80  PAS:=0.1:
B001   90  WHILE X<6 DO
B022  100  BEGIN
B022  110   WRITE('  ABS(',X:4:1,')=',ABS(X):3:1):
B06A  120   WRITE('  SQR(',X:4:1,')=',SQR(X):5:2);
B0B3  130   WRITE('  FRAC(',X:4:1,')=',FRAC(X):3:1):
B0FD  140   WRITELN('  EXP(',X:4:1,')=',EXP(X):7:3):
```

```
B149    PAS:=PAS+0.1;
B162    X:=X+PAS
B16B   END
B17C  END;
B185
B185  PROCEDURE PARAMETRUPOZITIV;
B188  BEGIN
B1A0   X:=0.2;
B1AD   PAS:=0.1;
B1BA   WHILE X<9 DO
B1DB   BEGIN
B1DB    WRITE('  SQRT(',X:3:1,')=',SQRT(X):8:6);
B225    WRITELN('  LN(',X*10:2:0,')=',LN(X*10):8:6);
B286    PAS:=PAS+0.1;
B29F    X:=X+PAS
B2A8   END
B2B9  END;
B2C2
B2C2  PROCEDURE TRIGONOMETRICE;
B2C5  BEGIN
B2DD   X:=0.2;
B2EA   PAS:=0.1;
B2F7   WHILE X<9 DO
B318   BEGIN
B318    WRITE('  SIN(',X:3:1,')=',SIN(X):9:6);
B361    WRITE('  COS(',X:3:1,')=',COS(X):9:6);
B3AA    WRITE('  TAN(',X:3:1,')=',TAN(X):10:6);
B3F3    WRITELN('  ARCTAN(',X:3:1,')=',180*ARCTAN(X)/3.1415:2:0);
B459    PAS:=PAS+0.1;
B472    X:=X+PAS
B47B   END
B48C  END;
B495
B495  BEGIN
B49E   WRITELN(CHR(16));
B4A8   ARITMETICE;WRITELN;
B4B0   PARAMETRUPOZITIV;WRITELN;
B4B8   TRIGONOMETRICE;WRITELN;
B4C0   WRITELN(CHR(16))
B4C7  END {$P}.
```

```
ABS(-2.1)=2.1  SQR(-2.1)= 4.41  FRAC(-2.1)=0.9  EXP(-2.1)=  0.122
ABS(-1.9)=1.9  SQR(-1.9)= 3.61  FRAC(-1.9)=0.1  EXP(-1.9)=  0.150
ABS(-1.6)=1.6  SQR(-1.6)= 2.56  FRAC(-1.6)=0.4  EXP(-1.6)=  0.202
ABS(-1.2)=1.2  SQR(-1.2)= 1.44  FRAC(-1.2)=0.8  EXP(-1.2)=  0.301
ABS(-0.7)=0.7  SQR(-0.7)= 0.49  FRAC(-0.7)=0.3  EXP(-0.7)=  0.497
ABS(-0.1)=0.1  SQR(-0.1)= 0.01  FRAC(-0.1)=0.9  EXP(-0.1)=  0.905
ABS( 0.6)=0.6  SQR( 0.6)= 0.36  FRAC( 0.6)=0.6  EXP( 0.6)=  1.822
ABS( 1.4)=1.4  SQR( 1.4)= 1.96  FRAC( 1.4)=0.4  EXP( 1.4)=  4.055
ABS( 2.3)=2.3  SQR( 2.3)= 5.29  FRAC( 2.3)=0.3  EXP( 2.3)=  9.974
ABS( 3.3)=3.3  SQR( 3.3)=10.89  FRAC( 3.3)=0.3  EXP( 3.3)= 27.112
ABS( 4.4)=4.4  SQR( 4.4)=19.36  FRAC( 4.4)=0.4  EXP( 4.4)= 81.450
ABS( 5.6)=5.6  SQR( 5.6)=31.36  FRAC( 5.6)=0.6  EXP( 5.6)=270.425

SQRT(0.2)=0.447213  LN( 2)=0.693147
SQRT(0.4)=0.632455  LN( 4)=1.386294
SQRT(0.7)=0.836660  LN( 7)=1.945909
SQRT(1.1)=1.048808  LN(11)=2.397894
SQRT(1.6)=1.264911  LN(16)=2.772588
SQRT(2.2)=1.483239  LN(22)=3.091041
SQRT(2.9)=1.702938  LN(29)=3.367295
SQRT(3.7)=1.923537  LN(37)=3.610916
SQRT(4.6)=2.144760  LN(46)=3.828641
SQRT(5.6)=2.366431  LN(56)=4.025349
SQRT(6.7)=2.588435  LN(67)=4.204691
SQRT(7.9)=2.810692  LN(79)=4.369447

SIN(0.2)= 0.198669  COS(0.2)= 0.980066  TAN(0.2)=  0.202707  ARCTAN(0.2)=11
SIN(0.4)= 0.389419  COS(0.4)= 0.921060  TAN(0.4)=  0.422793  ARCTAN(0.4)=22
SIN(0.7)= 0.644217  COS(0.7)= 0.764842  TAN(0.7)=  0.842286  ARCTAN(0.7)=35
SIN(1.1)= 0.891206  COS(1.1)= 0.453597  TAN(1.1)=  1.964756  ARCTAN(1.1)=48
SIN(1.6)= 0.999574  COS(1.6)=-0.029198  TAN(1.6)=-34.234137  ARCTAN(1.6)=58
SIN(2.2)= 0.808498  COS(2.2)=-0.588499  TAN(2.2)= -1.373830  ARCTAN(2.2)=66
SIN(2.9)= 0.239253  COS(2.9)=-0.970957  TAN(2.9)= -0.246407  ARCTAN(2.9)=71
SIN(3.7)=-0.529832  COS(3.7)=-0.848102  TAN(3.7)=  0.624728  ARCTAN(3.7)=75
SIN(4.6)=-0.993690  COS(4.6)=-0.112160  TAN(4.6)=  8.859647  ARCTAN(4.6)=78
SIN(5.6)=-0.631273  COS(5.6)= 0.775560  TAN(5.6)= -0.813955  ARCTAN(5.6)=80
SIN(6.7)= 0.404842  COS(6.7)= 0.914386  TAN(6.7)=  0.442748  ARCTAN(6.7)=82
SIN(7.9)= 0.998942  COS(7.9)=-0.045990  TAN(7.9)=-21.720232  ARCTAN(7.9)=83
```

d) Alte funcţii (existente în orice implementare)

In limbajul PASCAL există şi alte funcţii standard, unele regăsindu-se în toate implementările, altele fiind proprii lui HP4TM.

Funcţia SUCC

Forma generală a apelului funcţiei este: SUCC(X).

Parametrul X poate fi de orice tip ordinal şi returnează succesorul lui X în codul setului care defineşte tipul lui X.

Funcţia PRED

Forma generală a apelului funcţiei este: PRED(X).

Parametrul X poate fi de orice tip ordinal şi returnează predecesorul lui X în setul care defineşte tipul lui X.

Observaţie:

In cazul celor două funcţii, SUCC şi PRED, rezultatul funcţiei nu este definit dacă funcţiile se aplică ultimului, respectiv primului element din set. In implementarea HP4TM însă

PRED(primul element) = ultimul element,

SUCC(ultimul element) = primul element.

Exemplificarea funcţiilor PRED şi SUCC se poate urmări în programul **TESTPREDSUCC** (P.V.23).

```
AF7B  PROGRAM TESTPREDSUCC:
AF7B  CONST P=CHR(16);
AF7B  TYPE ENUM=(INTRARE.TRANSFER.ARITMETICE.ALTE);
AF7B  VAR C:CHAR;
AF84      LOGIC:BOOLEAN:
AF84      FUNCTIE:ENUM;
AF84
AF84  PROCEDURE SCRIECHAR:
AF87  BEGIN
AF9F   READLN;
AFA2   REPEAT
AFA2    READ(C):
AFAB    WRITE(P,'C=',C,' PRED(',C,')=',PRED(C)):
AFEE    WRITELN(' SUCC(',C,')=',SUCC(C));
B01C    WRITELN(P):
B024   UNTIL EOLN;
B02B   WRITELN;
B02E  END;
B034
B034  PROCEDURE SCRIU;
B037  BEGIN
B04F   WRITE('PRED(',LOGIC,')=',PRED(LOGIC),CHR(13));
B080   WRITELN('SUCC(',LOGIC,')=',SUCC(LOGIC));
B0AD  END;
B0B3
B0B3  PROCEDURE SCRIELOGIC;
B0B6  BEGIN
B0CE   LOGIC:=FALSE;
B0D2   SCRIU;
B0DB   LOGIC:=TRUE;
B0E0   SCRIU;
B0E9   WRITELN;
B0EC  END;
B0F2
B0F2  PROCEDURE SCRIEENUMERARE;
B0F5  BEGIN
B10D   FOR FUNCTIE:=INTRARE TO ALTE DO
B11C    BEGIN
B11F     WRITELN(ORD(SUCC(FUNCTIE)),'    ',ORD(PRED(FUNCTIE)
B147    END;
B156   WRITELN;
B159  END;
```

```
B15F  430
B15F  440 PROCEDURE SCRIEMAXINT;
B162  450 BEGIN
B17A  460  WRITE('PRED(',-MAXINT-1:6,')=',PRED(-MAXINT-1));
B1B5  470  WRITELN('SUCC(',MAXINT:5,')=',SUCC(MAXINT))
B1E8  480 END;
B1F1  490
B1F1  500 BEGIN
B1FA  510  SCRIECHAR;
B1FF  520  WRITE(P);
B204  530  SCRIELOGIC;
B209  540  SCRIEENUMERARE;
B20E  550  SCRIEMAXINT;
B213  560  WRITE(P)
B218  570 END {$P}.
```

```
C=A PRED(A)=@ SUCC(A)=B
C=> PRED(>)== SUCC(>)=?
C=- PRED(-)=, SUCC(-)=.
C=+ PRED(+)=* SUCC(+)=,
C=' PRED(')=& SUCC(')=(
C=! PRED(!)=  SUCC(!)="
C=_ PRED(_)=^ SUCC(_)=`
C=B PRED(B)=A SUCC(B)=C
C=a PRED(a)=` SUCC(a)=b
C=t PRED(t)=s SUCC(t)=u
PRED(FALSE)=TRUE
SUCC(FALSE)=TRUE
PRED(TRUE)=FALSE
SUCC(TRUE)=TRUE

1    255
2    0
3    1
4    2

PRED(-32768)=32767 SUCC(32767)=-32768
```

P.V.23.

Funcţia ODD

Forma generală a apelului funcţiei este: ODD(X).

Parametrul X trebuie să fie de tip întreg. Funcţia este de tip logic şi returnează TRUE dacă parametrul X este impar şi FALSE dacă parametrul X este par. Programul **TESTODD** (P.V.24) exemplifică utilizarea acestei funcţii.

```
ACCB   10 PROGRAM TESTODD;
ACCB   20 VAR INTREG:INTEGER;
ACD4   30
ACD4   40 PROCEDURE IMPARPAR;
ACD7   50 BEGIN
ACEF   60  FOR INTREG:=-3 TO 5 DO
AD0C   70   WRITELN('ODD(',INTREG:2,')=',ODD(INTREG):5)
AD4D   80 END;
AD59   90
AD59  100 BEGIN
AD62  110  WRITELN(CHR(16));WRITELN;
AD6F  120  IMPARPAR;
AD74  130  WRITELN;WRITELN(CHR(16))
AD7E  140 END {$P}.
```

```
ODD(-3)= TRUE
ODD(-2)=FALSE
ODD(-1)= TRUE
ODD( 0)=FALSE
ODD( 1)= TRUE
ODD( 2)=FALSE
ODD( 3)= TRUE
ODD( 4)=FALSE
ODD( 5)= TRUE
```

P.V.24.

E) Funcţii specifice implementării HP4TM

Funcţia RANDOM

Forma generală a apelului funcţiei este: RANDOM.

Funcţia returnează un număr pseudoaleator cuprins între 0 şi 255 inclusiv.

In programul **TESTRANDOM** (P.V.25), procedura VERIFICA realizează compararea celor 100 de numere generate prin metoda propusă în procedura SCRIERANDOM indicînd, dacă este cazul, egalitatea a două valori.

```
AE53   10 PROGRAM TESTRANDOM;
AE53   20 VAR I:INTEGER;
AE5C   30     A:REAL;
AE5C   40     B:ARRAY[1..1000]OF REAL;
AE5C   50     SW:BOOLEAN;
AE5C   60
AE5C   70 PROCEDURE SCRIERANDOM;
AE5F   80 BEGIN
AE77   90  FOR I:=1 TO 100 DO
AE91  100  BEGIN
AE94  110   A:=RANDOM/10E3*RANDOM/10E3/(RANDOM/10E3)*10;
AF01  120   IF A>1 THEN A:=A/10
AF38  130         ELSE
AF43  140          IF A<0.1 THEN
AF61  150             A:=A*10;
AF7A  160   B[I]:=A;
AFA7  170    WRITE(A:10:6);
AFBB  180   IF (I MOD 5)=0 THEN WRITELN;
AFD7  190  END;
AFDA  200  WRITELN
AFDA  210 END;
AFE3  220
AFE3  230 PROCEDURE VERIFICA;
AFE6  240 VAR J:INTEGER;
AFE6  250 BEGIN
AFFE  260  SW:=FALSE;
B002  270  FOR I:=2 TO 100 DO
B01C  280   FOR J:=1 TO I-1 DO
B047  290    IF B[J]=B[I]
B08E  300      THEN
B0A7  310      BEGIN
B0A7  320       SW:=TRUE;
B0AC  330       WRITELN(I,J,B[I])
B0E7  340      END
B0EA  350 END;
B0F8  360
B0F8  370 BEGIN
B101  380  WRITELN(CHR(16));WRITELN;
B10E  390  SCRIERANDOM;
B113  400  VERIFICA;
B118  410  IF NOT SW THEN WRITELN('TOATE DIFERITE!!!');
B140  420  WRITELN;WRITELN(CHR(16))
B14A  430 END {$P}.

0.427737  0.943255  0.542762  0.941071  0.124068
0.230459  0.221089  0.414285  0.818888  0.136888
0.147112  0.691257  0.167598  0.440023  0.130248
0.329424  0.779610  0.503913  0.689220  0.328926
0.257353  0.652167  0.142263  0.603077  0.146770
0.997301  0.700000  0.379518  0.119699  0.125336
0.212191  0.376096  0.101701  0.429474  0.560427
0.115489  0.418000  0.929999  0.132830  0.947872
0.744966  0.173282  0.552453  0.114088  0.149039
0.984046  0.484432  0.494118  0.198347  0.075455
0.728571  0.437400  0.030494  0.200686  0.051509
0.292280  0.274042  0.030000  0.564010  0.207278
0.352227  0.323926  0.168099  0.043056  0.151555
0.137020  0.029587  0.073280  0.171789  0.646923
0.101887  0.654084  0.229405  0.041216  0.671111
0.450079  0.106202  0.171826  0.932984  0.309661
0.139376  0.179294  0.201115  0.314210  0.936700
0.406779  0.376267  0.294366  0.101250  0.289420
0.367241  0.197200  0.247362  0.153000  0.232402
0.710181  0.017222  0.020887  0.388656  0.467478

TOATE DIFERITE!!!
```

P.V.25.

Funcţia ADDR

Forma generală a apelului funcţiei este: ADDR(V).

Funcţia este de tip întreg şi returnează adresa din memorie la care se află identificatorul desemnat de parametrul V care poate fi de orice tip.

Funcţia SIZE

Forma generală a apelului funcţiei este: SIZE(V).

Asemenea funcţiei ADDR, argumentul funcţiei SIZE este un identificator de variabilă. Funcţia returnează, ca şi în cazul anterior, un rezultat de tip întreg, care reprezintă numărul de octeţi ocupaţi în memorie de această variabilă.

In programul **ADDRSIZE** (P.V.26) au fost declarate variabile de tip întreg, real, logic, caracter, enumerat, mai multe tablouri unidimensionale de diferite tipuri şi un tablou bidimensional de elemente reale. In program sînt listate valorile funcţiilor ADDR şi SIZE pentru fiecare din variabilele declarate. Observăm că adresa la care a fost memorată variabila INTREG (#FF54) este mai mare decît adresa la care a fost memorată variabila REALUL (#FF50) cu toate că variabila INTREG a fost declarată înainte.

Imaginea zonei de memorie rezervată variabilelor:

SIRCHAR	SIRBOOL	MATRICE	SIR	TIP	CAR	BOOL	REALUL	INTREG
<-10->	<-10->	<-400->	<-20->	<-1->	<-1->	<-1->	<-4->	<-2->
#FD95	#FD9F	#FDA9	#FF39	#FF4D	#FF4E	#FF4F	#FF50	#FF54

Funcţiile ADDR şi SIZE se pot utiliza împreună pentru a salva pe casetă variabile, tablouri folosind procedura TOUT prezentată în paragraful 5.3.5. Astfel linia 270 din programul **ADDRSIZE** (P.V.26) are ca efect salvarea tabloului bidimensional MATRICE indicîndu-se numele, ('MATRICE ', de tip ARRAY[1..8] OF CHAR), adresa din memorie unde se află (ADDR(MATRICE)) şi lungimea (SIZE(MATRICE)).

```
PROGRAM ADDRSIZE:
TYPE ENUM=(INTREG.REALUL.BOOL.CAR);
VAR INTREG:INTEGER:
    REALUL:REAL:
    BOOL:BOOLEAN;
    CAR:CHAR:
    TIP:ENUM;
    SIR:ARRAY[1..10] OF INTEGER:
    MATRICE:ARRAY[1..10,1..10] OF REAL;
    SIRBOOL:ARRAY[1..10] OF BOOLEAN:
    SIRCHAR:ARRAY[1..10] OF CHAR;
BEGIN
 PAGE:
 WRITE(CHR(22).CHR(0),CHR(0));
 WRITELN(CHR(16)):WRITELN;
 WRITELN('ADDR(INTREG)=',ADDR(INTREG):4:H,'  SIZE(INTREG)=',SIZE(INTREG));
 WRITELN('ADDR(REALUL)=',ADDR(REALUL):4:H,'  SIZE(REALUL)=',SIZE(REALUL));
 WRITELN('ADDR(BOOL)=',ADDR(BOOL):4:H.'     SIZE(BOOL)=',SIZE(BOOL)):
 WRITELN('ADDR(CAR)=',ADDR(CAR):4:H,'      SIZE(CAR)=',SIZE(CAR));
 WRITELN('ADDR(TIP)=',ADDR(TIP):4:H,'      SIZE(TIP)=',SIZE(TIP));
 WRITELN('ADDR(SIR)=',ADDR(SIR):4:H,'      SIZE(SIR)=',SIZE(SIR));
 WRITELN('ADDR(MATRICE)=',ADDR(MATRICE):4:H,' SIZE(MATRICE)=',SIZE(MATRICE));
 WRITELN('ADDR(SIRBOOL)=',ADDR(SIRBOOL):4:H,' SIZE(SIRBOOL)=',SIZE(SIRBOOL));
 WRITELN('ADDR(SIRCHAR)=',ADDR(SIRCHAR):4:H,' SIZE(SIRCHAR)=',SIZE(SIRCHAR));
```

```
WRITELN;WRITELN(CHR(16));
TOUT('MATRICE ',ADDR(MATRICE),SIZE(MATRICE))
END {$P}.
```

```
ADDR(INTREG)=EA5D  SIZE(INTREG)=2
ADDR(REALUL)=EA59  SIZE(REALUL)=4
ADDR(BOOL)=EA58    SIZE(BOOL)=1
ADDR(CAR)=EA57     SIZE(CAR)=1
ADDR(TIP)=EA56     SIZE(TIP)=1
ADDR(SIR)=EA42     SIZE(SIR)=20
ADDR(MATRICE)=E8B2 SIZE(MATRICE)=400
ADDR(SIRBOOL)=E8A8 SIZE(SIRBOOL)=10
ADDR(SIRCHAR)=E89E SIZE(SIRCHAR)=10
```

P.V.26.

Funcţia PEEK

Forma generală a apelului funcţiei este: PEEK(X,T).

Primul parametru al acestei funcţii este de tip întreg şi specifică o adresă din memorie. Al doilea argument desemnează tipul rezultatului. Funcţia extrage informaţii din memorie de la adresa X conform tipului definit de T.

Astfel, dacă în memorie la adresa #EA60 (60000) introducem, folosind procedura POKE, informaţia 'PASCAL', atunci prin extragerea informaţiei de la această adresă, conform diferitelor tipuri, vom obţine informaţii diferite. In programul **EXPEEK** (P.V.27), în liniile 160, 170 au fost folosite posibilităţile oferite de funcţiile *PEEK*, *ORD*, *CHR* pentru a transforma prima treime a ecranului din INV VIDEO în TRUE VIDEO.

```
AE06   10 PROGRAM EXPEEK;
AE06   20 VAR I:INTEGER;
AE0F   30
AE0F   40 BEGIN
AE18   50  POKE(#EA60,'Pascal');
AE2E   60  PAGE;
AE33   70  WRITE(CHR(22),CHR(0),CHR(0));
AE48   80  WRITELN(CHR(16));WRITELN;
AE55   90  WRITELN(PEEK(#EA60,ARRAY[1..6] OF CHAR));
AE60  100  WRITELN(PEEK(#EA60,CHAR));
AE6A  110  WRITELN(PEEK(#EA60,INTEGER));
AE7A  120  WRITELN(PEEK(#EA60,REAL));
AE8B  130  WRITELN(PEEK(#EA60,BOOLEAN));
AE95  140  FOR I:=0 TO 5 DO
AEAF  150   WRITE(ORD(PEEK(#EA60+I,CHAR)):2:H,' ');
AED4  160  WRITELN(CHR(13),'     *** APASA O TASTA ***');
AF03  170  WRITELN;WRITELN(CHR(16));
AF10  180  REPEAT UNTIL INCH<>CHR(0);
AF25  190  FOR I:=0 TO 2047 DO
AF3F  200   POKE(16384+I,CHR(255-ORD(PEEK(16384+I,CHAR))))
AF71  210 END {$P}.
```

```
Pascal
P
24912
 2.46228E+29
TRUE
50 61 73 63 61 6C
     *** APASA O TASTA ***
```

P.V.27

Programul **DVMPBY2** (P.V.27a) reprezintă o aplicaţie interesantă privind folosirea funcţiilor standard PEEK şi ADDR. Programul realizează înmulţirea, respectiv împărţirea cu 2 a unui număr real. Pentru realizarea înmulţirii (împărţirii) se foloseşte reprezentarea internă a unui număr real (vezi paragraful 5.3.3).

```
AECE  { PROGRAMUL DEMONSTREAZA CUM SE FOLOSESC
AECE     FUNCTIILE STANDARD
AECE         PEEK
AECE         ADDR
AECE  SI PROCEDURA POKE
AECE    PENTRU MODIFICAREA VARIABILELOR PASCAL
AECE
AECE  PROGRAM DVMPBY2;
AECE  VAR r:REAL;
AED7      i:INTEGER;
AED7
AED7  FUNCTION DV2(x:REAL):REAL;
AEDA  BEGIN
AEF2   i:=ADDR(x)+1;                  {adresa exponent}
AEFD   POKE(i,PRED(PEEK(i,CHAR)));  {decrementeaza exponent}
AF08   DV2:=x
AF08  END;
AF2A
AF2A  FUNCTION MP2(x:REAL):REAL;
AF2D  BEGIN
AF45   i:=ADDR(x)+1;                  {adresa exponent}
AF50   POKE(i,SUCC(PEEK(i,CHAR)));  {incrementeaza exponent}
AF5B   MP2:=x
AF5B  END;
AF7D
AF7D  BEGIN
AF86   REPEAT
AF86    WRITE('Introdu numarul r=');
AFA6    READ(r);
AFB0    WRITELN(CHR(16));
AFBA    WRITELN(r:7:2,' impartit cu 2 este ',DV2(r):7:2);
B00F    WRITELN(r:7:2,' inmultit cu 2 este ',MP2(r):7:2);
B064    WRITELN;WRITELN(CHR(16))
B06E   UNTIL r=0
B082  END {$P}.
```

```
 345.55 impartit cu 2 este  172.77
 345.55 inmultit cu 2 este  691.10


 -13.89 impartit cu 2 este   -6.94
 -13.89 inmultit cu 2 este  -27.78


   0.02 impartit cu 2 este    0.01
   0.02 inmultit cu 2 este    0.04


  99.00 impartit cu 2 este   49.50
  99.00 inmultit cu 2 este  198.00


-466.00 impartit cu 2 este -233.00
-466.00 inmultit cu 2 este -932.00


   0.00 impartit cu 2 este    0.00
   0.00 inmultit cu 2 este    0.00
```

P.V.27a

Funcţia INP

Forma generală a apelului funcţiei este: INP(P).

Această funcţie se foloseşte pentru a avea acces direct la porturile de intrare ale microprocesorului Z80, fără a mai utiliza procedura INLINE. P este un parametru de tip întreg, valoarea lui fiind încărcată în dublul registru BC. Funcţia furnizează un rezultat de tip caracter care se obţine prin executarea instrucţiunii Z80: IN A, (C).

5.4.3. Funcţii definite în program

Flexibilitatea limbajului este asigurată de posibilitatea folosirii funcţiilor standard precum şi a celor proprii, definite de utilizator.

Reamintim că o funcţie este un subprogram care calculează şi returnează subprogramului apelant o singură valoare. In PASCAL valoarea calculată este în mod obligatoriu de *tip simplu* sau *referinţă* (pointer).

După cum se observă din diagrama de sintaxă din fig.5.7, funcţia poate să nu conţină nici un parametru. In cazul în care există parametri, sintaxa lor este identică cu cea de la proceduri. De altfel, toate precizările referitoare la parametrii formali şi efectivi şi domeniul de valabilitate sînt valabile şi în cazul funcţiilor.

Deci deosebirea între funcţie şi procedură constă în faptul că o funcţie returnează o singură valoare, în timp ce procedura poate să returneze mai multe valori sau niciuna. Apelul de funcţie este un operand într-o expresie, iar apelul de procedură este o instrucţiune.

Pentru exemplificare considerăm programul **MEDIEINMATRICE** (P.V.28) care rezolvă aceeaşi problemă ca şi P.V.18, respectiv calculează media elementelor maxime de pe fiecare linie a unei matrici. Atît procedura ELMAX cît şi procedura MEDIE din P.V.18 calculează o singură valoare, elementul maxim de pe o linie, respectiv media elementelor dintr-un vector. Se poate pune problema înlocuirii acestor proceduri cu nişte funcţii. Liniile 130-210 conţin în acest caz corpul funcţiei ELMAX. In declararea funcţiei apar doi parametri: I de tip întreg (reprezintă indicele de linie) şi A de tip MATRICE (tip definit în program). In linia 200 avem

```
ELMAX:=MAX;
```

în care numele funcţiei (fără parametri) apare în stînga atribuirii. O astfel de atribuire trebuie să apară în fiecare funcţie definită. In acest fel valoarea atribuită se va transmite prin intermediul numelui funcţiei în blocul apelant.

Observaţie:

In locul variabilei MAX nu se putea folosi direct numele funcţiei ELMAX, deoarece apariţia numelui funcţiei în dreapta unei instrucţiuni de atribuire sau într-o instrucţiune de control este echivalentă cu un apel recursiv, caz în care trebuie specificaţi şi parametrii actuali.

Liniile 230-300 descriu funcţia VALMED. Se observă totala identitate a corpului funcţiei cu corpul procedurii MEDIE din P.V.18., cu excepţia liniilor 210, respectiv 230.

```
210 PROCEDURE MEDIE (VAR V:LINIE;VAR VALMED:REAL);
230 FUNCTION VALMED (V:LINIE):REAL;
```

In primul caz (linia 210 din P.V.18), valoarea calculată se transmite prin intermediul parametrului VALMED, transmis prin referinţă.

In al doilea caz (linia 230 din P.V.28), valoarea calculată este transmisă prin intermediul funcţiei VALMED.

```
B01C  { PROGRAMUL CALCULEAZA MEDIA ELEMENTELOR MAXIME
B01C    DE PE FIECARE LINIE A UNEI MATRICI            }
B01C
B01C  PROGRAM MEDIEINMATRICE;
B01C  CONST N=5;
B01C  TYPE LINIE=ARRAY[1..N] OF REAL;
B01C       MATRICE=ARRAY[1..N,1..N] OF REAL;
B01C  VAR X:REAL;
B025      VECTOR:LINIE;
B025      MAT:MATRICE;
B025      I,J:1..N;
B025
B025  FUNCTION ELMAX(I:INTEGER;A:MATRICE):REAL;
B028  VAR J:1..N;
B028      MAX:REAL;
B028  BEGIN
B040   MAX:=A[I,1];
B092   FOR J:=2 TO N DO
B0B5    IF MAX<A[I,J] THEN MAX:=A[I,J];
B176   ELMAX:=MAX
B176  END;
B19D
B19D  FUNCTION VALMED(V:LINIE):REAL;
B1A0  VAR I:1..N;
B1A0      X:REAL;
B1A0  BEGIN
B1B8   X:=0;
B1CA   FOR I:=1 TO N DO X:=X+V[I];
B23A   VALMED:=X/N
B250  END;
B26C
B26C  BEGIN
B275   FOR I:=1 TO N DO
B28F    BEGIN
B292     FOR J:=1 TO N DO
B2AC      BEGIN
B2AF       READ(X);
B2B9       MAT[I,J]:=X
B2F1      END;
B305     WRITELN;
B308
B308  { *** apel functie ELMAX *** }
B308
B308     VECTOR[I]:=ELMAX(I,MAT)
B33B    END;
B34F   PAGE;
B354   WRITELN(CHR(16));WRITELN;
B361   FOR I:=1 TO N DO
B37B    BEGIN
B37E     FOR J:=1 TO N DO
B398      WRITE(MAT[I,J]:6:2);
B3EA     WRITELN
B3EA    END;
B3F0   WRITELN;
B3F3   WRITELN('ELEMENTELE MAXIME ALE LINIILOR');
B41F   FOR J:=1 TO N DO WRITE(VECTOR[J]:6:2);
B46F   WRITELN;WRITELN;
B475
B475  { *** apel functie VALMED *** }
B475
B475   WRITELN('MEDIA ELEMENTELOR MAXIME:  ',VALMED(VECTOR):6:2);
B4C4   WRITELN;WRITELN(CHR(16))
B4CE  END {$P}.
```

```
  5.00 77.00 89.00 -8.00  5.00
 55.00 23.00 -7.00 -5.00  0.00
 -1.00  6.00 88.00 45.00 88.00
 76.00 12.00 -3.00 77.00  6.00
 56.00 76.00 -4.00  0.00  1.00

ELEMENTELE MAXIME ALE LINIILOR
 89.00 55.00 88.00 77.00 76.00

MEDIA ELEMENTELOR MAXIME:  77.00
```

Observaţie:

In versiunea HP4TM, procedurile şi funcţiile nu sînt acceptate ca parametri, lucru permis în alte implementări ale limbajului. In aceste implementări transmiterea parametrilor funcţie sau procedură se realizează specificînd parametrul respectiv prin antetul său. La apel, antetului îi corespunde un nume de funcţie sau procedură utilizator, anterior definite.

Exemplu:

```
PROGRAM PARAMFUNCTIE;
  VAR A1,B1,A2,B2:REAL;
      N1,N2:INTEGER;
  FUNCTION TG(X:REAL):REAL;
    BEGIN
      TG:=TAN(X);
    END;
  FUNCTION SINCOS(X:REAL):REAL;
    BEGIN
      SINCOS:=SIN(X)+COS(X);
    END;
  FUNCTION INTEGRALA(FUNCTION F(X:REAL):REAL;A,B:REAL;
                            N:INTEGER):REAL;
    VAR L,S:REAL;
        I:INTEGER;
      BEGIN
        S:=0;
        L:=(B-A)/N;
        X:=A;
        FOR I:=1 TO N DO
          BEGIN
            S:=S+F(X)*L;
            X:=X+L
          END
      END;
BEGIN
  READLN(A1,B1,N1);
  WRITELN(INTEGRALA(SINCOS,A1,B1,N1));
  READLN(A2,B2,N2);
  WRITELN(INTEGRALA(TG,A2,B2,N2))
END.
```

5.4.4. Stocarea datelor în memorie

Utilizatorul limbajului PASCAL are nevoie de informaţii privind modul de stocare a variabilelor în timpul executării programului, stocare care diferă pentru:

a) variabile globale - declarate în blocul principal al programului;

b) variabile locale - declarate într-un bloc intern;

c) parametri şi valori returnate - transmise către, sau de la proceduri şi funcţii.

a) Variabile globale

Variabilelor globale le este alocat un spaţiu în stiva de execuţie, începînd de la adrese mari spre adrese mici. Astfel, dacă la execuţie adresa stivei este #B000, iar variabilele globale ale programului sînt declarate:

```
VAR I:INTEGER;
   CH:CHAR;
    X:REAL;
```

atunci

I (care ocupă 2 octeţi) va fi stocat la #B000-2 şi #B000-1 adică #AFFE şi #AFFF;

CH (care ocupă un octet) va fi stocat la #AFFE-1 adică la #AFFD;

X (care ocupă 4 octeţi) va fi stocat la #AFF9,#AFFA,#AFFB, #AFFC;

(vezi şi paragraful 5.4.3. programul **ADDRSIZE** (P.V.26)).

b) Variabile locale

Adresa de început a zonei alocate variabilelor locale aparţinînd unui bloc se păstrează în zona a cărei adresă se obţine scăzînd 4 din conţinutul lui IX, adică la IX-4.

Exemplu:

```
PROCEDURE TEST;
  VAR I,J:INTEGER;
```

I-(întreg pe 2 octeţi) va fi plasat la adresele IX-4-2 şi IX-4-1, adică la IX-6,IX-5;

J-(întreg pe 2 octeţi) va fi plasat la IX-8 şi IX-7.

c) Parametri şi valori returnate

Parametrii transmişi prin valoare sînt trataţi la fel ca variabilele locale. Cu cît un parametru este declarat mai devreme, cu atît adresa de memorie la care este stocat este mai mare. Spre deosebire însă de variabilele locale, la IX+2 este fixată adresa cea mai mică şi nu adresa cea mai mare.

Exemplu:

```
PROCEDURE TEST(I:REAL;J:INTEGER);
```

J-(i se alocă primele două locaţii) este stocat la IX+2 şi IX+3;

I-(i se alocă patru locaţii) este stocat la IX+4, IX+5, IX+6, IX+7.

Pentru parametrii transmişi prin adresă se copiază adresele lor pe stivă (se vor ocupa 2 octeţi pentru fiecare adresă).

Exemplu:

```
PROCEDURE TEST(I:INTEGER;VAR X:REAL);
```

-adresa lui X este conţinută în IX+2 şi IX+3;

-valoarea lui I este conţinută în IX+4 şi IX+5.

Valorile returnate de funcţii sînt plasate pe stivă deasupra primului parametru.

Exemplu:

```
FUNCTION TEST(I:INTEGER):REAL;
```

-I se găseşte la IX+2 şi IX+3;

-pentru valoarea returnată se rezervă spaţiu la IX+4, IX+5, IX+6 şi IX+7.

In finalul acestui paragraf vom exemplifica modul de dezvoltare *descendent* al programelor PASCAL, rezolvînd problema ordonării crescătoare a unui vector A.

Bineînţeles, în practică problema nu apare în această formă simplă. In general, elementele vectorului pot fi formate dintr-un set de informaţii (de exemplu: nume, clasă, medie etc), cu alte cuvinte putem avea un vector de articole. Unul din cîmpurile articolului (numit *cheie*) are un rol aparte, ordonarea făcîndu-se după valoarea acestui cîmp.

In funcţie de locul în care sînt păstrate elementele vectorului A în timpul prelucrării, distingem două tipuri de sortări:

-*sortare internă*: elementele lui A sînt păstrate în memoria internă a calculatorului;

-*sortare externă*: elementele lui A sînt păstrate pe un suport extern.

In mod evident, metodele de sortare diferă în funcţie de tipul sortării (internă sau externă).

In cazul sortării interne, există o multitudine de strategii de sortare, fiecare avînd avantajele şi dezavantajele sale care sînt analizate în funcţie de diverse criterii:

-memoria ocupată;
-număr de comparaţii;
-număr de deplasări ale elementelor;
-timp de execuţie.

Luînd în considerare aceste criterii, s-a observat (statistic) că metoda care dă cele mai bune rezultate este metoda descrisă în 1962 de C.A.R.Hoare, numită *metodă de sortare rapidă (quicksort)*, pe care o vom descrie pe scurt în continuare.

Vom folosi vectorul A cu elemente numere întregi. Prin parcurgerea vectorului pornind de la ambele capete (pe rînd) şi interschimbarea elementelor care nu sînt în relaţia cerută, se împarte vectorul în două părţi, nu neapărat de lungime egală, cu proprietatea că toate elementele din prima parte sînt mai mici (sau mai mari) decît toate elementele din cea de-a doua parte. Unul din subvectori este memorat (prin indicii de început şi de sfîrşit), iar cu cel rămas se procedează analog. Subşirurile memorate sînt prelucrate apoi pe rînd în acelaşi mod, în ordinea inversă memorării lor.

In programul **SORTAREQUICK** (P.V.29q) vom da o implementare nerecursivă a acestui algoritm, folosind ca stivă un tablou în care se memorează în ordine primul şi ultimul element al vectorului care trebuie păstrat.

```
B25A  PROGRAM SORTAREQUICK;
B25A  CONST N=10;
B25A  TYPE ELEMENT=INTEGER;
B25A       SIR=ARRAY[1..N]OF ELEMENT:
B25A       INDICE=0..N:
B25A  VAR A:SIR;
B263
B263  PROCEDURE QUICKSORT;
B266  CONST D=100;
B266  VAR STIVA:ARRAY[1..D] OF INDICE;
B266      VIRF:INTEGER;
B266      PRIM,ULTIM,I,J:INDICE;
B266      MIJLOC:ELEMENT;
B266
B266      PROCEDURE PUSH(V:INDICE);
B269      BEGIN
B281        VIRF:=VIRF+1;
B29E        STIVA[VIRF]:=V;
B2D4      END;
```

```
B2DB
B2DB        FUNCTION POP:INDICE;
B2DE        BEGIN
B2F6          POP:=STIVA[VIRF];
B32A          VIRF:=VIRF-1
B346        END;
B34D
B34D        PROCEDURE SWAP(I1,I2:INDICE);
B350        VAR AUX:ELEMENT;
B350        BEGIN
B368          AUX:=A[I1];
B38F          A[I1]:=A[I2];
B3D3          A[I2]:=AUX
B3F1        END;
B407
B407    BEGIN
B41F      VIRF:=0;
B42F      PUSH(1);PUSH(N);
B441      REPEAT
B441        ULTIM:=POP;PRIM:=POP;
B46C        REPEAT
B46C          I:=PRIM;J:=ULTIM;
B49F          MIJLOC:=A[(PRIM+ULTIM) DIV 2];
B4ED          REPEAT
B4ED            WHILE A[I]<MIJLOC DO I:=I+1;
B54E            WHILE A[J]>MIJLOC DO J:=J-1;
B5AB            IF I<=J THEN
B5D0                         BEGIN
B5D0                           SWAP(I,J);
B5ED                           I:=I+1;
B606                           J:=J-1;
B61F                         END
B61F          UNTIL I>J;
B642          IF I<ULTIM THEN
B666                          BEGIN
B666                            PUSH(I);
B677                            PUSH(ULTIM);
B688                          END;
B688          ULTIM:=J;
B6A0        UNTIL PRIM>ULTIM;
B6C3      UNTIL VIRF=0;
B6DD    END;
B6E8
B6E8    PROCEDURE CITESTESIR;
B6EB    VAR I:INDICE;
B6EB    BEGIN
B703      FOR I:=1 TO N DO
B726        BEGIN
B729          WRITE('A[',I:2,']=');
B752          READ(A[I]);
B778        END;
B77B    END;
B782
B782    PROCEDURE SCRIESIR;
B785    VAR I:INDICE;
B785    BEGIN
B79D      FOR I:=1 TO N DO WRITE(A[I]);
B7ED      WRITELN;
B7F0    END;
B7F7
B7F7    BEGIN
B800      WRITELN('SIRUL ARE ',N:2,' ELEMENTE');
B838      WRITELN('INTRODU ELEMENTELE SIRULUI:');
B861      CITESTESIR;
B866      WRITELN('SIRUL INITIAL ESTE:');
B887      SCRIESIR;
B88C      WRITELN;
B88F      QUICKSORT;
B894      WRITELN('SIRUL ORDONAT ESTE:');
B8B5      SCRIESIR;
B8BA    END {$P}.
```

P.V.29q

Procedura QUICKSORT conţine la rîndul ei blocul a trei subprograme:

- -PUSH pentru memorarea în stivă a unei valori;
- -POP pentru extragerea din stivă a unei valori;
- -SWAP pentru interschimbarea a două valori.

In program avem o singură variabilă globală: tabloul A, iar variabilele STIVA, VIRF, PRIM, ULTIM, I, J, MIJLOC

sînt locale procedurii QUICKSORT, deci sînt globale pentru procedurile PUSH, POP, SWAP şi necunoscute pentru procedurile CITESTESIR şi SCRIESIR.

Variabila AUX este locală procedurii SWAP iar variabila I este locală procedurilor CITESTESIR, SCRIESIR.

Intre variabila I din procedura CITESTESIR şi variabila I din procedura SCRIESIR nu se poate produce nici un incident deoarece fiecare este cunoscută doar în procedura în care este declarată şi necunoscută în exterior.

5.5. RECURSIVITATE

5.5.1. Generalităţi

Deşi conceptul de recursivitate este cunoscut de mult în matematică, în programare a fost introdus odată cu apariţia unor limbaje de nivel înalt (ALGOL, PASCAL).

Despre un obiect se spune că este recursiv dacă este definit prin el însuşi.

Exemple:

1)-mulţimea numerelor naturale
- a) 1 este număr natural;
- b) succesorul unui număr natural este un număr natural

2)-funcţia factorial (*n*!) -definită pentru numere naturale
- a) 0!=1;
- b) dacă n>0, atunci n!=n*(n-1)!

3)-structura de tip arbore
- a) un singur nod (O) este un arbore (arbore vid);
- b) dacă *a1* şi *a2* sînt arbori, atunci structura următoare este un arbore:

```
      0
     / \
   a1   a2
```

Puterea recursivităţii rezidă în posibilitatea de a defini un set infinit de obiecte printr-o relaţie sau un set finit de relaţii.

Recursivitatea nu trebuie confundată cu *iteraţia*, deşi ele sînt strîns legate:

-*iteraţia* înseamnă execuţia repetată a unei porţiuni de program, pînă în momentul în care este îndeplinită o anumită condiţie; fiecare execuţie se duce pînă la capăt, se verifică îndeplinirea condiţiei şi în cazul unui anumit răspuns se reia execuţia de la început (de exemplu: structurile repetitive WHILE, REPEAT, FOR);

-*recursivitatea* presupune, de asemenea, execuţia repetată a unei porţiuni de program, dar, spre deosebire de iteraţie, în cadrul recursivităţii condiţia de terminare este verificată în cursul execuţiei programului (nu la sfîrşit) şi în cazul unui anumit rezultat, întreaga porţiune de program (care formează blocul unei proceduri sau funcţii) este apelată din nou ca orice subprogram, în particular ca procedură a blocului care *încă nu şi-a terminat execuţia*; în momentul satisfacerii condiţiei de terminare, se reia execuţia programului apelant

exact din punctul în care s-a apelat pe el însuşi; acest lucru este valabil pentru toate apelurile anterioare satisfacerii condiţiei.

Subprogramele recursive necesită evaluarea unei *condiţii de terminare*, fără de care un apel recursiv ar conduce la un ciclu infinit.

In programare, recursivitatea este exprimată prin proceduri sau funcţii autoapelate. Dacă o procedură P conţine o referinţă directă la ea însăşi, se spune că este *direct recursivă*; dacă P conţine o referinţă la o altă procedură Q, care la rîndul ei conţine o referinţă (directă sau indirectă) la P, se spune că P este *indirect recursivă*. Programul **SPATII** (P.V.39) este un exemplu pentru ilustrarea recursivităţii indirecte.

De regulă, unui subprogram i se asociază un set de obiecte locale subprogramului (constante, etichete, tipuri, variabile, proceduri, funcţii) care sînt definite în subprogram şi care nu există sau nu au sens în afara lui. Ori de cîte ori un astfel de subprogram este apelat recursiv, se creează un nou set de astfel de variabile locale, specifice apelului curent (alocîndu-li-se spaţiu pe stiva de execuţie). Deşi aceste variabile au acelaşi nume ca şi cele corespunzătoare lor din apelul anterior al subprogramului (în calitate de program apelant), ele au valori distincte şi orice conflict de nume este evitat prin regulile care stabilesc domeniul de existenţă al identificatorilor. Identificatorii se referă întotdeauna la setul de variabile cel mai recent creat. Aceleaşi reguli sînt valabile şi pentru parametrii de apel ai subprogramului care sînt asociaţi prin definiţie setului de variabile.

In general, recursivitatea se foloseşte doar atunci cînd "adîncimea" recursivităţii (numărul apelurilor recursive generat de un anumit apel) este relativ mică (ea fiind în mod obligatoriu finită), deoarece fiecare apel recursiv al subprogramului necesită alocarea unui volum de memorie variabilelor sale curente. In plus, alături de aceste variabile, trebuie memorată şi starea curentă a programului, cu scopul de a fi refăcută atunci cînd noua activare a subprogramului se termină şi urmează ca activitatea anterioară să fie reluată. In concluzie, dacă în execuţia unui program recursiv numărul apelurilor recursive este relativ mare, se măreşte mult numărul valorilor care trebuie memorate şi programul devine ineficient.

5.5.2. Program recursiv de ilustrare a mecanismului recursivităţii

Considerăm un şir de cuvinte urmate de cîte un blanc. Se cere afişarea cuvintelor, aşa cum au fost ele date, precum şi inversate.

In programul **INVERSARE** (P.V.29) procedura recursivă INVERSEAZA citeşte cîte un caracter şi îl afişează imediat prin intermediul variabilei locale Z de tip caracter. Dacă ultimul caracter citit nu este blanc, procedura se autoapelează, în caz contrar se afişează caracterul Z.

Pînă la apariţia primului blanc, execuţia procedurii are ca efect tipărirea caracterelor în ordinea în care au fost citite. Fiecare autoapel presupune salvarea în stiva sistemului

a contextului apelului, care în cazul de faţă este doar variabila locală Z.

Apariţia primului blanc presupune oprirea apelurilor recursive ale procedurii, declanşînd continuarea execuţiei procedurii pentru fiecare din apelurile anterioare, din punctul următor apelului pînă la terminarea sa. In cazul de faţă aceasta înseamnă tipărirea caracterului memorat în variabila Z. Acest şir de reveniri va produce la început tipărirea unui blanc, după care *sînt tipărite caracterele în ordinea inversă citirii lor*, deoarece fiecare terminare a execuţiei procedurii determină revenirea în apelul anterior, revenire care presupune reactualizarea contextului apelului. Deoarece contextele sînt salvate pe stivă, reactualizarea lor are loc în ordine inversă celei în care au fost memorate.

```
AD67  PROGRAM INVERSARE;
AD67
AD67  (  INVERSAREA UNOR CUVINTE
AD67            exemplu de
AD67   A P E L   R E C U R S I V     )
AD67
AD67  CONST EOL=CHR(13);
AD67  VAR I,N:INTEGER;
AD70
AD70  PROCEDURE INVERSEAZA;
AD73  VAR Z:CHAR;
AD73  BEGIN
AD8B   READ(Z); WRITE(CHR(16),Z,CHR(16));
ADA5   IF Z<>' ' THEN INVERSEAZA;
ADBE   WRITE(CHR(16),Z,CHR(16))
ADD2  END;
ADD9
ADD9  BEGIN
ADE2   READ(N);
ADE8   FOR I:=1 TO N DO
AE06    BEGIN
AE09     INVERSEAZA;
AE0E     WRITE(CHR(16),EOL,CHR(16))
AE21    END
AE21  END (*P*).
```

```
INVERSARE  ERASREVNI

TREI  IERT
DOI  IOD
PATRU  URTAP
CINCI  ICNIC



INVERSARE

ERASREVNI


TREI  IERT


DOI


IOD
```

P.V.29.

In programul **INVERSARE** nu există nici o secvenţă de schimbare a elementelor şi nu s-a declarat nici un tablou de caractere. Totuşi, datorită apelurilor recursive, utilizînd valorile actuale de pe stiva de execuţie se obţine cuvîntul inversat. Modul de lucru al programului pentru inversarea cuvîntului 'DOI' se poate urmări în schema din fig.5.8.

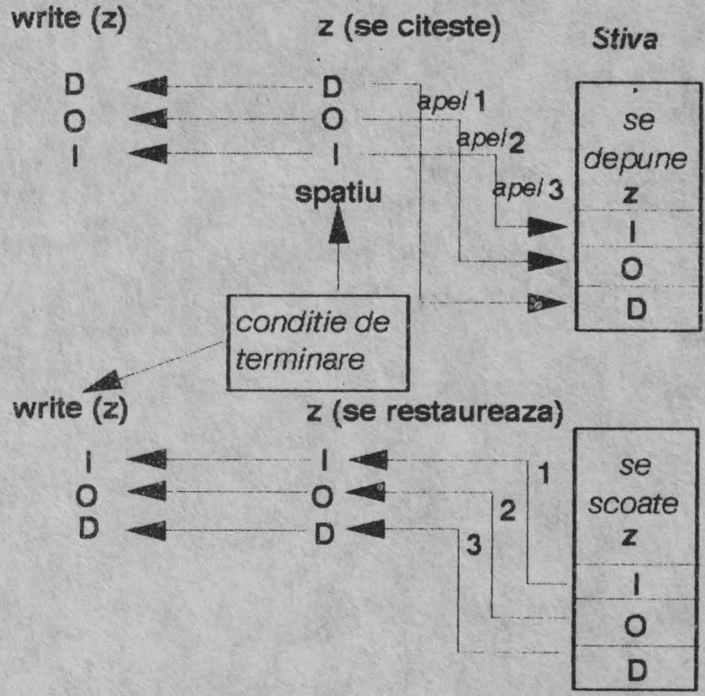


fig.5.8

5.5.3. Utilizarea recursivităţii

Algoritmii recursivi sînt potriviţi a fi utilizaţi în principiu în două cazuri:

- cînd *problema* este definită recursiv,
- cînd *datele* care trebuie prelucrate sînt definite în *termeni recursivi*.

Cu toate acestea, o formulare recursivă a problemei nu justifică întotdeauna utilizarea unui algoritm recursiv.

In cele ce urmează vom prezenta două probleme care, cu toate că se enunţă în termeni recursivi, se rezolvă mai eficient iterativ.

1) Calculul factorialului

Un exemplu clasic în care se cere calculul unor valori care se definesc cu ajutorul unor relaţii recursive îl reprezintă factorialul unui număr, care, aşa cum s-a văzut în paragraful 5.3.1., se defineşte astfel:

a) 0!=1;

b) dacă n>0, atunci n!=n*(n-1)!

Se observă că oricare element, cu excepţia primului, este definit folosind termenul anterior.

In programul **FACTREC** (P.V.30) este utilizat acest mod de calcul al factorialului, folosind o funcţie recursivă.

```
AD5B   PROGRAM FACTREC;
AD5B
AD5B   {Algoritm RECURSIV pentru calculul
AD5B                FACTORIALULUI        }
AD5B
```

```
AD5B   CONST EOL=CHR(13);
AD5B   VAR I:INTEGER;
AD64
AD64   FUNCTION FACT(N:INTEGER):INTEGER;
AD67   BEGIN
AD7F    IF N<0 THEN FACT:=-1
AD9A     ELSE
ADA5      IF (N=0) OR (N=1) THEN FACT:=1
ADD2       ELSE FACT:=N*FACT(N-1)
ADEA   END;
AE05
AE05   BEGIN
AE0E    WRITELN(CHR(16));WRITELN;
AE1B    FOR I:=-6 TO 7 DO
AE38     WRITE(I:2,'!=',FACT(I),EOL);
AE6D    WRITELN;WRITELN(CHR(16))
AE77   END {$P}.

-6!=-1
-5!=-1
-4!=-1
-3!=-1
-2!=-1
-1!=-1
 0!=1
 1!=1
 2!=2
 3!=6
 4!=24
 5!=120
 6!=720
 7!=5040
```

P.V.30

Observaţie:

Factorialul se defineşte pentru numere naturale. In program, dacă funcţia este apelată pentru un număr negativ, anomalia se semnalează atribuind factorialului valoarea -1, urmată de abandonarea execuţiei funcţiei.

Este evident că pentru calculul factorialului recursivitatea poate fi înlocuită printr-o simplă iteraţie, aşa cum se poate vedea în programul **FACTITER** (P.V.31).

```
AD3A   PROGRAM FACTITER;
AD3A   CONST P=CHR(16);
AD3A   VAR N,F:INTEGER;
AD43    { N - NUMAR CITIT }
AD43    { F - VALOARE FACTORIALA }
AD43
AD43   FUNCTION FAC(X:INTEGER):INTEGER;
AD46   VAR I,F:INTEGER;
AD46   BEGIN
AD5E    IF X<0 THEN FAC:=-1
AD79           ELSE
AD84            BEGIN
AD84             F:=1;
AD8D             FOR I:=2 TO X DO F:=F*I;
ADD5              FAC:=F
ADD5            END
ADE1   END;
ADEA
ADEA   BEGIN
ADF3    REPEAT
ADF3     READ(N);
ADFC     WRITELN(P,'N=',N,'  N!=',FAC(N),CHR(13),P)
AE44    UNTIL FALSE
AE49   END {$P}.
N=3    N!=6
N=6    N!=720
N=0    N!=1
N=-5    N!=-1
N=7    N!=5040
N=1    N!=1
```

P.V.31

2) Problema generării numerelor lui Fibonacci

Această problemă se bazează de asemenea pe o definiţie recursivă, totuşi se tratează mult mai eficient cu ajutorul iteraţiei. Numerele lui Fibonacci, sînt definite prin următoarea relaţie recursivă:

FIB(0)=0, FIB(1)=1,

FIB(n+1)=FIB(n)+FIB(n-1), pentru n>0.

Această definiţie recursivă conduce la algoritmul de calcul din programul **FIBOREC** (P.V.32a).

```
AEC0   PROGRAM FIBOREC;
AEC0
AEC0   {Algoritm RECURSIV pentru calculul
AEC0   sirului lui FIBONACCI             }
AEC0
AEC0   CONST EOL=CHR(13);
AEC0   VAR N:INTEGER;
AEC9
AEC9   FUNCTION FIBO(N:INTEGER):INTEGER;
AECC   BEGIN
AEE4    IF N=0 THEN FIBO:=0 ELSE
AF05    IF N=1 THEN FIBO:=1 ELSE
AF26       FIBO:=FIBO(N-1)+FIBO(N-2)
AF44   END;
AF62
AF62   BEGIN
AF6B    FOR N:=0 TO 23 DO
AF85     WRITE('FIBO(',N+1:2,')=',FIBO(N),EOL)
AFC8   END {$P}.
```

P.V.32a

Din păcate apelul recursiv utilizat în această situaţie conduce la o manieră ineficientă de calcul, deoarece numărul de apeluri creşte exponenţial odată cu *n*. De exemplu, pentru n=5 sînt necesare 15 apeluri, conform fig.5.9.

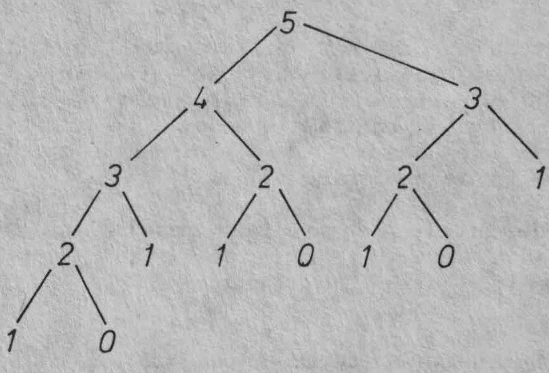


fig. 5.9

Este evident că un astfel de program nu este practic şi că numerele lui Fibonacci pot fi calculate cu ajutorul unei scheme iterative care elimină recalcularea valorilor deja calculate. Problema este rezolvată iterativ în programul **FIBOITER** (P.V.32b).

```
AEC0   PROGRAM FIBOITER;
AEC0   VAR I:INTEGER;
AEC9
AEC9   {Algoritm ITERATIV pentru calculul
AEC9   sirului lui FIBONACCI             }
AEC9
AEC9   FUNCTION FIB(N:INTEGER):INTEGER;
AECC   VAR X,Y,Z:INTEGER;
```

```
AECC  BEGIN
AEE4   IF N=0 THEN FIB:=0 ELSE
AF05    BEGIN
AF05     I:=1; X:=1;Y:=0;
AF1D     WHILE I<N DO
AF37      BEGIN
AF37       I:=I+1;Z:=X;X:=X+Y;Y:=Z
AF64      END;
AF73     FIB:=X
AF73    END
AF7F  END;
AF8B
AF8B  BEGIN
AF94   FOR I:=0 TO 23 DO
AFAE    WRITE(CHR(16),'FIBO(',I+1:2,')=',FIB(I),CHR(13),CHR(16))
B001  END {$P}.

FIBO( 1)=0
FIBO( 2)=1
FIBO( 3)=1
FIBO( 4)=2
FIBO( 5)=3
FIBO( 6)=5
FIBO( 7)=8
FIBO( 8)=13
FIBO( 9)=21
FIBO(10)=34
FIBO(11)=55
FIBO(12)=89
FIBO(13)=144
FIBO(14)=233
FIBO(15)=377
FIBO(16)=610
FIBO(17)=987
FIBO(18)=1597
FIBO(19)=2584
FIBO(20)=4181
FIBO(21)=6765
FIBO(22)=10946
FIBO(23)=17711
FIBO(24)=28657
```

P.V.32b

Eficienţa celui de-al doilea mod de calcul al numerelor lui Fibonacci se observă clar urmărind timpul de calcul al primelor 20 de numere. In cazul algoritmului iterativ numerele sînt afişate practic instantaneu, pe cînd în cazul algoritmului recursiv pe măsura creşterii lui *n*, timpul de calcul devine din ce în ce mai mare.

Din considerentele de mai sus rezultă că *utilizarea recursivităţii trebuie evitată ori de cîte ori există o rezolvare mai eficientă, bazată pe iteraţie.*

Observaţie:

Majoritatea programelor recursive pot fi transformate în programe iterative, dar pe de altă parte recursivitatea are domenii bine stabilite, în care se aplică cu succes. In general se apreciază că algoritmii a căror natură este recursivă trebuie formulaţi ca proceduri recursive, uşurînd în acest fel activitatea de programare.

5.5.4. Tipuri de algoritmi recursivi

In cele ce urmează vom prezenta cîţiva algoritmi recursivi, grupaţi în mai multe categorii:

A)-algoritmi care implementează metode de rezolvare definite recursiv;

B)-algoritmi de divizare; ("divide et impera")
C)-algoritmi pentru aflarea tuturor soluţiilor unei probleme;
D)-algoritmi cu revenire; ("backtracking")
E)-algoritmi care necesită recursivitate indirectă;
F)-algoritmi care prelucrează structuri de date definite recursiv (exemple în capitolul VI).

A. Algoritmi care implementează metode de rezolvare definite recursiv

Algoritmul pentru determinarea celui mai mare divizor comun (CMMDC) a două numere naturale, cunoscut sub numele de "algoritmul lui Euclid" poate fi descris recursiv în modul următor:

a)-dacă unul dintre numere este nul, atunci CMMDC al lor este celălalt număr;
b)-dacă nici unul din numere nu este nul, atunci CMMDC coincide cu CMMDC al numărului mai mic şi al restului împărţirii numărului mai mare la cel mai mic.

Programul **EUCLID** (P.V.33) urmăreşte definiţia recursivă de mai sus. Prin rularea lui se pot urmări pe rînd valorile parametrilor M şi N, pînă cînd unul devine zero.

```
AD56   10 PROGRAM EUCLID;
AD56   20 VAR A,B,X:INTEGER;
AD5F   30
AD5F   40 FUNCTION CMMDC(M,N:INTEGER):INTEGER;
AD62   50 BEGIN
AD7A   60  WRITELN('M=',M,'N=',N);
ADAF   70  IF N=0 THEN CMMDC:=M ELSE    CMMDC:=CMMDC(N,M MOD N)
ADED   80 END;
AE07   90
AE07  100 BEGIN
AE10  110  WRITE('A=');READ(A);
AE23  120  WRITE('B=');READ(B);
AE36  130  WRITELN(CHR(16));WRITELN;
AE43  140  X:=CMMDC(A,B);
AE55  150  WRITE(CHR(13),'CMMDC(',A,CHR(8),',',B,CHR(8),')=',X,CHR(13));
AEAF  160  WRITELN;WRITELN(CHR(16))
AEB9  170 END {$P}.

M=456 N=234
M=234 N=222
M=222 N=12
M=12 N=6
M=6 N=0

CMMDC(456  ,234  )=6
```

P.V.33

B. Algoritmi de divizare

Una din metodele fundamentale de proiectare a algoritmilor se bazează pe *tehnica divizării* (*divide et impera*). Ideea de bază constă în a descompune (a divide) problema complexă în mai multe subprobleme a căror rezolvare este mai simplă, soluţiile acestora contribuind la determinarea soluţiei problemei iniţiale.

1) Algoritm pentru aflarea minimului şi maximului unui vector

Este evident că această problemă se poate rezolva uşor printr-un proces iterativ care parcurge întregul vector, comparînd fiecare element cu cel mai mare, respectiv cel mai mic element determinat pînă la momentul comparării curente.

Pentru un vector cu *n* elemente numărul de comparaţii este 2n-2 (numărul de comparaţii reprezintă unul din criteriile de clasificare a eficienţei unui algoritm). Se observă că fiecare element este supus la două comparări. Acest inconvenient poate fi evitat utilizînd *tehnica divizării*: se imparte vectorul în două subşiruri şi se compară minimele şi maximele acestor subşiruri, obţinîndu-se minimul şi maximul absolut. In continuare se procedează analog cu cele două subşiruri, de fiecare dată reducîndu-se la jumătate dimensiunea subşirurilor. Pentru dimensiunile 1 sau 2 soluţia este imediată.

Metoda descrisă este implementată în programul **MINMAX** (P.V.34).

```
AF37  PROGRAM MINMAX;
AF37  CONST N=10;
AF37  TYPE TAB=ARRAY[1..N] OF INTEGER;
AF37  VAR A:TAB;
AF40      I,INF,SUP:INTEGER;
AF40
AF40  PROCEDURE DOMENIU(A:TAB;I,J:INTEGER;VAR MIN,MAX:INTEGER);
AF43  VAR MIJLOC,MIN1,MAX1,MIN2,MAX2:INTEGER;
AF43  BEGIN
AF5B   IF J<=I+1 THEN
AF77    BEGIN
AF77     IF A[I]<A[J] THEN
AFCF      BEGIN
AFCF       MIN:=A[I];
B000       MAX:=A[J]
B029      END
B031       ELSE
B034      BEGIN
B034       MIN:=A[J];
B065       MAX:=A[I]
B08E      END
B096    END
B096       ELSE
B099    BEGIN
B099     MIJLOC:=(I+J) DIV 2;
B0BB     DOMENIU(A,I,MIJLOC,MIN1,MAX1);
B0F7     DOMENIU(A,MIJLOC+1,J,MIN2,MAX2);
B134     IF MAX1>MAX2 THEN MAX:=MAX1 ELSE MAX:=MAX2;
B174     IF MIN1<MIN2 THEN MIN:=MIN1 ELSE MIN:=MIN2
B1AA    END
B1B5  END;
B1C4
B1C4  BEGIN
B1CD   PAGE;
B1D2   WRITELN('Introdu elementele sirului:');
B1FB   FOR I:=1 TO N DO
B215    BEGIN
B218     WRITE('A[',I:2,']=');READ(A[I])
B261    END;
B264   WRITE(CHR(16));
B26B   WRITELN;WRITELN('Sirul este:');
B287   FOR I:=1 TO N DO WRITE(A[I]:3);
B2CE   WRITELN;WRITELN;
B2D4   DOMENIU(A,1,N,INF,SUP);
B2FA   WRITELN('      INF=',INF,'    SUP=',SUP,CHR(13));
B33E   WRITELN;WRITE(CHR(16))
B348  END {$P}.

Sirul este:
  6 -9  0  3 55  1 61 -8  7 13

          INF=-9    SUP=61
```

P.V.34.

Deşi conceperea acestui program este mai dificilă decît în cazul algoritmului iterativ, din punctul de vedere al numărului de comparaţii este mai rapid. In literatura de specialitate se indică pentru acest algoritm un număr de 3n/2-2 comparaţii, deci mai mic decît în cazul anterior.

2) Algoritm pentru determinarea permutărilor primelor *n* numere naturale

Principiul pe care se bazează algoritmul, folosind tehnica divizării, este următorul: *pentru a obţine permutările de n elemente este suficient să fixăm pe rînd cîte un element şi să permutăm celelalte n-1 elemente.* Procedînd analog pentru cele n-1 elemente, se ajunge la permutări de 1 element (cazul banal).

Programul **PERMUTARE** (P.V.35) realizează calculul permutărilor primelor 4 numere naturale. Numerele se presupun memorate în tabloul A[I],I=1,2,3,4. Procedura PERMUTA, de parametru K, verifică dacă s-a ajuns la permutări banale (K=1), caz în care se afişează tabloul A, reprezentînd o permutare. In cazul K<>1, fiecare element al tabloului este adus pe poziţia K şi se apelează procedura pentru K-1 elemente. După revenire se reface starea iniţială schimbînd din nou elementele din poziţiile I şi K.

Condiţia K=1 reprezintă condiţia de terminare care limitează adîncimea apelului recursiv, determinînd revenirea la apelurile anterioare.

```
AE75  PROGRAM PERMUTARE;
AE75
AE75  { OBTINEREA PERMUTARILOR DE N ELEMENTE FOLOSIND
AE75         A P E L U L   R E C U R S I V
AE75    se fixeaza un element si se obtin permutarile
AE75    pentru celelalte N-1 elemente ...              }
AE75
AE75  CONST N=4;CR=CHR(13);PR=CHR(16);
AE75  VAR I:INTEGER;
AE7E      A:ARRAY[1..N] OF 1..9;
AE7E
AE7E  PROCEDURE TIPARESTE;
AE81  VAR I:INTEGER;
AE81  BEGIN
AE99   FOR I:=1 TO N DO WRITE(A[I]);
AEE9   WRITE(CR)
AEEE  END;
AEF5
AEF5  PROCEDURE PERMUTA(K:INTEGER);
AEF8  VAR I,X:INTEGER;
AEF8  BEGIN
AF10   IF K=1 THEN TIPARESTE ELSE
AF31    BEGIN
AF31     FOR I:=1 TO K DO
AF5B      BEGIN
AF5E       X:=A[I];A[I]:=A[K];A[K]:=X;
AFF2       PERMUTA(K-1);
B003       X:=A[I];A[I]:=A[K];A[K]:=X
B08C      END
B097    END
B09B  END;
B0A4
B0A4  BEGIN
B0AD   FOR I:=1 TO N DO A[I]:=I;
B0F0   WRITE(PR);
B0F5   PERMUTA(N);
B0FE   WRITE(CR,PR)
B108  END {$P}.
```

```
2 3 4 1
3 2 4 1
3 4 2 1
4 3 2 1
2 4 3 1
4 2 3 1
4 3 1 2
3 4 1 2
3 1 4 2
1 3 4 2
4 1 3 2
1 4 3 2
```

```
2 4 1 3
4 2 1 3
4 1 2 3
1 4 2 3
2 1 4 3
1 2 4 3
2 3 1 4
3 2 1 4
3 1 2 4
1 3 2 4
2 1 3 4
1 2 3 4
```

P.V.35

C. Algoritmi pentru determinarea tuturor soluţiilor unei probleme

1) Generarea partiţiilor lui N în părţi egale cu 1 sau 2

Să se determine în cîte moduri se poate tăia o bucată de material, astfel încît fiecare parte să aibă lungimea 1 sau 2, ştiind că bucata iniţială are lungimea N, exprimată printr-un număr întreg.

Exemplu:

pentru N=3 există 3 posibilităţi:

a) 1,1,1
b) 1,2
c) 2,1

Algoritmul se bazează pe următoarea tehnică de lucru:

-pentru N>1 există două posibilităţi:

1)-la început se taie o bucată de lungime 1 şi bucata rămasă (de lungime N-1) se taie în toate modurile posibile;

2)-la început se taie o bucată de lungime 2 şi bucata rămasă (de lungime N-2) se taie în toate modurile posibile;

-pentru N=1, avem o bucată de lungime 1;

-pentru N=0, nu există nici o posibilitate de tăiere.

In programul **TAIETURA** (P.V.36) lungimile bucăţilor tăiate sînt vizualizate prin semnele '.'(însemnînd o bucată de lungime 1) şi '-'(semnificînd lungime 2). Acest mod de reprezentare grafică a tăieturilor necesită utilizarea unui tablou de caractere în care se depun pe rînd caracterele corespunzătoare tăieturilor.

Condiţia de terminare o reprezintă atingerea dimensiunii 0 sau 1, cînd se afişează soluţia găsită.

```
ADEC   10 PROGRAM TAIETURA:
ADEC   20 VAR I,K,X:INTEGER:
ADF5   30     Z:ARRAY[1..9] OF CHAR;
ADF5   40
ADF5   50 PROCEDURE P(N:INTEGER);
ADF8   60 BEGIN
AE10   70  IF N>1 THEN
AE26   80   BEGIN
AE26   90    K:=K+1;
AE2D  100    Z[K]:='.';P(N-1);
AE5B  110    Z[K]:='-';P(N-2);
AE8A  120    K:=K-1
AE92  130   END
AE91  140  ELSE
```

```
AE94      BEGIN
AE94        FOR I:=1 TO K DO WRITE(Z[I]);
AED5        IF N=1 THEN WRITE('.');
AEEF        WRITE(CHR(13))
AEF6      END
AEF6    END;
AEFD
AEFD    BEGIN
AF06     WRITELN(CHR(16)):WRITELN;
AF13     FOR X:=1 TO 8 DO
AF2D      BEGIN
AF30        PAGE;
AF35        WRITE(CHR(22),CHR(0),CHR(0));
AF4A        WRITELN('BUCATA ARE LUNGIMEA: ',X);
AF75        K:=0;
AF7B        WRITE(CHR(13));
AF82        P(X);
AF8B        REPEAT UNTIL INCH<>CHR(0);
AFA0        WRITELN;
AFA3      END;
AFA6     WRITELN;WRITELN(CHR(16))
AFB0    END {$P}.
```

```
BUCATA ARE LUNGIMEA:1

.

BUCATA ARE LUNGIMEA:2

..
-

BUCATA ARE LUNGIMEA:3

...
.-
-.

BUCATA ARE LUNGIMEA:4

....
..-
.-.
-..
--

BUCATA ARE LUNGIMEA:5

.....
...-
..-.
.-..
.--
-...
-.-
--.

BUCATA ARE LUNGIMEA:6

......
....-
...-.
..-..
..--
.-...
.-.-
.--.
-....
-..-
-.-.
--..
---

BUCATA ARE LUNGIMEA:7

.......
.....-
....-.
...-..
```

```
...--
..-...
..-.-
..--.
.-....
.-..-
.-.-.
.--..
.---
-.....
-...-
-..-.
-.-..
-.--
--...
--.-
---.

BUCATA ARE LUNGIMEA:8

........
......-
.....-.
....-..
....--
...-...
...-.-
...--.
..-....
..-..-
..-.-.
..--..
..---
.-.....
.-...-
.-..-.
.-.-..
.-.--
.--...
.--.-
.---.
-......
-....-
-...-.
-..-..
-..--
-.-...
-.-.-
-.--.
--....
--..-
--.-.
---..
----
```

P.V.36.

Studiind programul se observă maniera de lucru: în momentul cînd în şir se depune caracterul '.', deci bucata curentă are lungimea 1, se apelează procedura pentru N-1, iar cînd în şir se depune '-', corespunzător unei bucăţi de lungime 2, se apelează procedura pentru N-2.

2) Determinarea tuturor posibilităţilor de ieşire dintr-un labirint

In cadrul algoritmului, labirintul este reprezentat sub forma unui tablou de caractere de dimensiuni (N+1, N+1). In programul **LABIRINT** (P.V.37) procedura POKELABIRINT formează un labirint de dimensiuni 7*7 ai cărui "pereţi" au fost marcaţi prin caracterul "*", iar culoarele prin ' '. In program există şi posibilitatea introducerii de către utilizator a unui labirint propriu, această variantă figurînd în programul P.V.37 sub formă de comentariu. Prin anularea acoladelor şi eliminarea apelului procedurii POKELABIRINT, fiecare utilizator îşi poate introduce datele dorite. De asemenea, modificarea dimensiunilor labirintului se poate realiza simplu, modificînd constanta N.

Drumul de ieşire din labirint este determinat cu ajutorul procedurii CAUTA căreia i se transmit ca şi parametri coordonatele punctului curent (X,Y), ele fiind în acelaşi timp indici ai tabloului care materializează labirintul. Punctul de pornire este centrul labirintului, dar el poate fi oricare alt punct din interior (caz în care se modifică parametrii din apelul procedurii CAUTA). Căutarea se face în felul următor:

- dacă elementul curent este ' ', atunci el se marchează cu '.', fiind un element al unui posibil drum de ieşire;
- dacă s-a ajuns la una din margini, rezultă că s-a găsit un drum de ieşire din labirint şi se afişează tabloul;
- în caz contrar se apelează recursiv procedura CAUTA pentru cele patru puncte din vecinătatea imediată a celui curent, deci pentru punctele (X+1,Y), (X,Y+1), (X-1,Y), (X,Y-1).

Deoarece traseul este marcat cu ajutorul caracterului '.', parcurgerea unui drum deja ales nu este posibilă. Este de asemenea important că marcajul de drum se şterge imediat ce s-a ajuns într-o fundătură. Ştergerea se execută prin punerea în tablou a unui caracter ' 'pe poziţia curentă, înainte de părăsirea procedurii.

```
B061  PROGRAM LABIRINT;
B061  CONST N=6;
B061  VAR I,J,SOL:INTEGER;
B06A      M:ARRAY[0..N,0..N] OF CHAR;
B06A
B06A  PROCEDURE TIPARIRE;
B06D  BEGIN
B085   WRITE(CHR(22),CHR(0),CHR(0));
B09A   SOL:=SOL+1;
B0A1   WRITELN(CHR(16));
B0AB   WRITELN;WRITELN('Solutia:',SOL);WRITELN;
B0D0   FOR I:=0 TO N DO
B0EA    BEGIN
B0ED     WRITE('          ');
B100     FOR J:=0 TO N DO WRITE(M[I,J]);
B157     WRITELN
B157    END;
B15D   WRITELN;WRITELN(CHR(16))
B167  END;
B170
B170  PROCEDURE CAUTA(X,Y:INTEGER);
B173  BEGIN
B18B   IF M[X,Y]=' ' THEN
B1D3    BEGIN
B1D3     M[X,Y]:='.';
B211     IF ((X MOD N)=0) OR ((Y MOD N)=0) THEN
B248      BEGIN
B248       TIPARIRE;
B251       REPEAT UNTIL INCH<>CHR(0)
B25C      END
B266        ELSE
B269      BEGIN
B269       CAUTA(X+1,Y);
B281       CAUTA(X,Y+1);
B299       CAUTA(X-1,Y);
B2B1       CAUTA(X,Y-1)
B2C0      END;
B2C9     M[X,Y]:=' '
B306    END
B307  END;
B311
B311  PROCEDURE POKELABIRINT;
B314
B314  {  *** generare LABIRINT ***
B314         se poate modifica   }
B314
B314  BEGIN
B32C   POKE(ADDR(M[0]),'*******');
B35E   POKE(ADDR(M[1]),'*   * *');
B390   POKE(ADDR(M[2]),'  *   *');
B3C2   POKE(ADDR(M[3]),'* *  **');
B3F4   POKE(ADDR(M[4]),'*    **');
B426   POKE(ADDR(M[5]),'** *   ');
B458   POKE(ADDR(M[6]),'** ****');
B48A  END;
```

```
B490
B490   BEGIN
B499   {  *** introducere LABIRINT ***
B499    READLN;
B499    FOR I:=0 TO N DO
B499     BEGIN
B499       FOR J:=0 TO N DO READ(M[I,J]);
B499       READLN;
B499     END;}
B499
B499    PAGE;
B49E    POKELABIRINT;
B4A3    SOL:=-1;
B4AC    TIPARIRE;
B4B1    REPEAT UNTIL INCH<>CHR(0);
B4C6    CAUTA(N DIV 2,N DIV 2)
B4DE   END {$P}.
```

```
Solutia:0

        *******
        *   * *
          *   *
        * *  **
        *    **
        ** *
        ** ****


Solutia:1

        *******
        *   * *
          *   *
        * *. **
        *  ..**
        ** *...
        ** ****


Solutia:2

        *******
        *...* *
         .*.. *
        *.*..**
        *....**
        **.*
        **.****


Solutia:3

        *******
        *...* *
        ..*.. *
        * *..**
        *  ..**
        ** *
        ** ****


Solutia:4

        *******
        *   * *
          *   *
        * *. **
        * .. **
        **.*
        **.****
```

Solutia:5

```
*******
*...* *
 .*.. *
*.*..**
*....**
** *...
** ****
```

Solutia:6

```
*******
*   * *
..*   *
*.*. **
*... **
** *
** ****
```

P.V.37.

D) Algoritmi cu revenire

Aceşti algoritmi se bazează pe ideea găsirii soluţiilor unor probleme, nu pe baza unui set fix de reguli de calcul, ci prin încercări repetate şi reveniri în caz de nereuşită. Modalitatea de realizare a acestei tehnici constă în descompunerea obiectivului în obiective parţiale. De obicei acestea sînt exprimate în termeni recursivi şi constau în prelucrarea unui număr finit de situaţii. Obţinerea unor soluţii parţiale sau finale care nu satisfac cerinţele, provoacă revenirea recursivă în cadrul procesului de calcul pînă la obţinerea soluţiei dorite, motiv pentru care aceşti algoritmi se numesc *algoritmi cu revenire (backtracking)*.

Problema celor opt regine

Un exemplu binecunoscut de utilizare a algoritmilor cu revenire îl reprezintă problema celor 8 regine. Această problemă poate fi rezolvată prin încercări, necesitînd o mare cantitate de muncă, răbdare, precizie,- atribute în care calculatorul excelează, depăşind omul chiar atunci cînd acesta este un geniu. (Pînă în prezent nu a fost găsită o soluţie analitică completă a acestei probleme, deşi cercetarea ei a fost începută în 1850 de către C.F. GAUSS, care însă nu a rezolvat-o complet.)

Problema este următoarea: să se dispună pe tabla de şah 8 regine, astfel incît nici una dintre ele să nu le atace pe celelalte, ţinînd cont de regulile jocului de şah.

Prima problemă care apare este accea a reprezentării poziţiilor celor 8 regine pe tabla de şah. Ideea de a reprezenta tabla de şah cu ajutorul unei matrici de 8*8 este cea mai simplă, dar, din păcate, conduce la calcule greoaie şi complicate pentru determinarea cîmpurilor disponibile.

Intrucît pe fiecare coloană L este plasată o singură regină (1<=L<=I), se poate face următoarea alegere de structurare a datelor:

```
VAR X:ARRAY[1..8] OF INTEGER;
    A:ARRAY[1..8] OF BOOLEAN;
    B:ARRAY[2..16] OF BOOLEAN;
    C:ARRAY[0..14] OF BOOLEAN;
```

cu următoarele semnificaţii:

-X[I] precizează poziţia reginei în coloana I;

-A[J]=TRUE precizează că nici o regină nu ameninţă linia J

-B[K]=TRUE precizează că nici o regină nu ameninţă diagonala secundară K (/);

-C[K]=TRUE precizează că nici o regină nu ameninţă diagonala principală K (\).

Procedura recursivă de plasare a unei regine arată astfel:

```
PROCEDURA INCEARCA(I:INTEGER);
  BEGIN
   {INITIALIZEAZA SELECTIA POZITIILOR PENTRU A I-A REGINA}
   REPEAT
   {SELECTEAZA POZITIA URMATOARE}
     IF SIGURA THEN
       BEGIN
         {PUNE REGINA}
         IF I<8 THEN
           BEGIN
             INCEARCA(I+1);
               IF NEREUSITA THEN {IA REGINA}
           END
       END
   UNTIL REUSITA OR {NU MAI EXISTA POZITII}
  END;
```

Este clar că această procedură exprimată cu ajutorul comentariilor trebuie explicitată cu ajutorul instrucţiunilor.

In programul **REGINE** (P.V.38), nu se vor reprezenta poziţiile reginelor pe tabla de şah, ci faptul că o regină a fost sau nu plasată de-a lungul fiecărei linii sau diagonale. Din regulile jocului de şah se ştie că regina ameninţă coloana, linia şi cele două diagonale care conţin cîmpul pe care se află. Rezultă deci că fiecare coloană va putea conţine o singură regină. In acest fel alegerea poziţiei celei de-a I-a regine poate fi restrînsă numai la coloana I. Parametrul I reprezintă numărul coloanei, iar procesul de selecţie se restrînge la una din cele 8 valori posibile ale indicelui J care precizează linia.

Se ştie că pe o tablă de şah există 15 diagonale principale (\) şi 15 diagonale secundare (/) - cîte una de lungime maximă, dintr-un colţ în colţul opus, celelalte paralele cu acestea.

Caracteristica unei diagonale secundare (/) este aceea că suma coordonatelor I şi J este constantă, iar a unei diagonale principale (\), că diferenţa este constantă.

Folosind aceste reprezentări şi observaţii, comentariile din procedura INCEARCA se pot explicita astfel:

```
{PUNE REGINA} în poziţia (I,J) devine:
    X[I]:=J;
    A[J]:=FALSE;
    B[I+J]:=FALSE;        {2<=I+J<=16}
    C[I-J+7]:=FALSE;      {0<=I-J+7<=14}
```

{IA REGINA} devine:

A[J]:=TRUE;
B[I+J]:=TRUE;
C[I-J+7]:=TRUE;

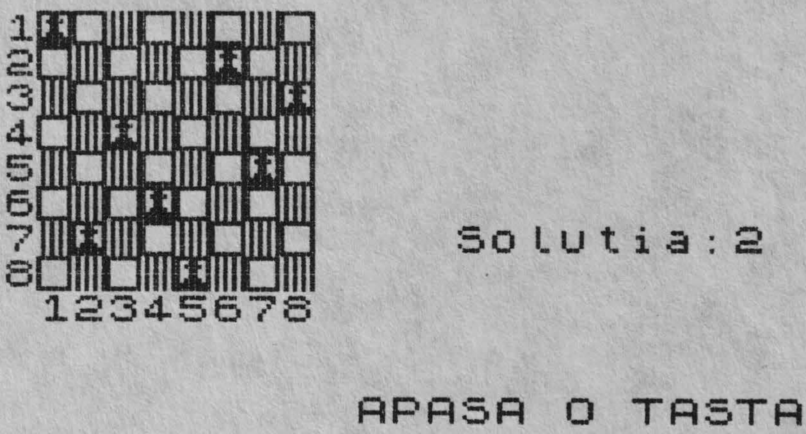
Condiţia SIGURA este îndeplinită dacă destinaţia (I,J) aparţine unei linii şi unor diagonale libere (TRUE), ceea ce poate fi exprimat astfel: A[J] AND B[I+J] AND C[I-J+7].

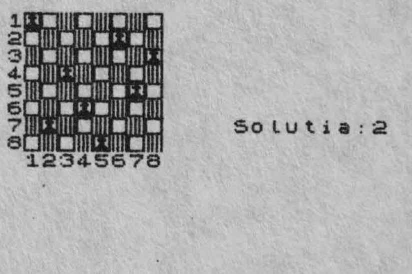
Pentru vizualizarea celor 92 de soluţii pe monitor (în implementarea HP4TM), în program a mai fost folosit un tablou de caractere Y:ARRAY[1..8] OF CHAR, care are rolul de a memora cîte o linie a tablei de şah. De asemenea pentru reprezentarea grafică a tablei de şah şi a reginelor, a fost folosită procedura UDG.

Programul afişează pe ecran toate cele 92 soluţii, dar este de remarcat faptul că există doar 12 soluţii distincte, celelalte putîndu-se obţine prin rotirea tablei de şah.

```
B19C  PROGRAM REGINE;
B19C  VAR I,SOL:INTEGER;
B1A5      A:ARRAY[1..8] OF BOOLEAN;
B1A5      B:ARRAY[2..16] OF BOOLEAN;
B1A5      C:ARRAY[0..14] OF BOOLEAN;
B1A5      X:ARRAY[1..8] OF INTEGER;
B1A5      Y:ARRAY[1..8] OF CHAR;
B1A5
B1A5  PROCEDURE TIPARIRE;
B1A8  VAR I,K:INTEGER;
B1A8      C:CHAR;
B1A8  BEGIN
B1C0   WRITE(CHR(22),CHR(0),CHR(0));WRITELN;
B1D8   SOL:=SOL+1;
B1DF   FOR K:=1 TO 8 DO
B202    BEGIN
B205     I:=1;
B20E     WHILE I<8 DO
B228      BEGIN
B228       Y[I]:=CHR(144);
B24A       Y[I+1]:=CHR(146);
B26D       I:=I+2
B278      END;
B27E     IF ODD(K)=TRUE  THEN
B295      BEGIN
B295       C:=Y[8];
B2B1       FOR I:=8 DOWNTO 2 DO
B2D3        Y[I]:=Y[I-1];
B314       Y[1]:=C
B32D      END;
B332     Y[X[K]]:=CHR(145);
B36F     WRITE(K:1);
B37E     FOR I:=1 TO 8 DO WRITE(Y[I]);
B3C6     WRITELN
B3C6    END;
B3CC   WRITE(' 12345678');
B3E0   WRITE(CHR(22),CHR(7),CHR(13),'Solutia:',SOL);
B411   WRITE(CHR(22),CHR(21),CHR(10),'APASA O TASTA');
B43E   REPEAT UNTIL INCH<>CHR(0);
B453   IF INCH IN ['S'] THEN
B47A     TOUT('SCR-REG ',16384,6912)
B497  END;
B4A0
B4A0  PROCEDURE INCEARCA(I:INTEGER);
B4A3  VAR J:INTEGER;
B4A3  BEGIN
B4BB   FOR J:=1 TO 8 DO
B4DE   IF A[J] AND B[I+J] AND C[I-J+7] THEN
B568    BEGIN
B568     X[I]:=J;
B591     A[J]:=FALSE;B[I+J]:=FALSE;
B5E0     C[I-J+7]:=FALSE;
B617     IF I<8 THEN INCEARCA(I+1)
B636      ELSE TIPARIRE;
B64B     A[J]:=TRUE;B[I+J]:=TRUE;
B69C     C[I-J+7]:=TRUE
B6D3    END
B6D4  END;
```

```
B6DF
B6DF  PROCEDURE UDG;
B6E2  BEGIN
B6FA   POKE(23675,23296);
B706   POKE(23296,255);
B712   FOR I:=0 TO 5 DO POKE(23297+I,129);
B749   POKE(23303,255);
B755   POKE(23304,255);
B761   POKE(23305,153);
B76D   POKE(23306,189);
B779   POKE(23307,153);
B785   POKE(23308,153);
B791   POKE(23309,189);
B79D   POKE(23310,255);
B7A9   POKE(23311,255);
B7B5   FOR I:=0 TO 7 DO POKE(23312+I,85)
B7E5  END;
B7F2
B7F2  BEGIN
B7FB   UDG;SOL:=0;
B806   FOR I:=1 TO 8 DO A[I]:=TRUE;
B844   FOR I:=2 TO 16 DO B[I]:=TRUE;
B883   FOR I:=0 TO 14 DO C[I]:=TRUE;
B8C0   PAGE;
B8C5    INCEARCA(1)
B8C9  END {$P}.
```





P.V.38.

E. Algoritmi care necesită recursivitate indirectă

Dacă o procedură se autoapelează, ea se numeşte *direct recursivă*, iar dacă această autoapelare se realizează prin intermediul altor proceduri, ea se numeşte *indirect recursivă*.

Programul SPATII (P.V.39) prelucrează un şir de caractere al cărui sfîrşit este marcat de caracterul '$'. Se cere eliminarea spaţiilor suplimentare din acest şir, adică reducerea numărului de spaţii, care separă două cuvinte adiacente, la unul singur şi suprimarea spaţiilor iniţiale şi finale.

Fie A şirul (nevid) dat şi B şirul rezultat, posibil identic cu A, dar care nu mai conţine caracterul final '$'.

Se vor utiliza trei proceduri şi anume:

-procedura SPINIT, are ca scop eliminarea spaţiilor iniţiale din acest text; ea se autoapelează pînă la detectarea unui caracter diferit de spaţiu sau '$'; dacă caracterul care urmează nu este '$', caracterul respectiv se depune în şirul B şi se apelează procedura CUVINT care nu a fost încă definită; se impune deci declararea procedurii CUVINT utilizînd directiva FORWARD înainte de definirea procedurii SPINIT;

-procedura SPATIU tratează spaţiile dintre cuvinte, reducînd numărul acestora la unul singur, pe care îl depune în şirul B; dacă următorul caracter tratat este diferit de spaţiu sau '$', se apelează procedura CUVINT declarată anterior cu FORWARD;

-procedura CUVINT tratează caracterele din cadrul unui cuvînt şi le depune în şirul B; dacă caracterul tratat este spaţiu se apelează procedura SPATIU; orice alt caracter (diferit de spaţiu sau de '$') va provoca autoapelarea procedurii CUVINT.

Se observă că fiecare procedură în parte este recursivă direct şi procedurile împreună folosesc o recursivitate indirectă prin intermediul procedurii CUVINT.

Evident, problema poate fi rezolvată mult mai simplu nerecursiv, dar am ales această tehnică din motive didactice.

```
B0F5   PROGRAM SPATII;
B0F5   VAR I,J:0..79;
B0FE       A,B:ARRAY[0..79] OF CHAR;
B0FE
B0FE   PROCEDURE CUVINT;FORWARD;
B101
B101   PROCEDURE SPINIT;
B104     { elimina spatiile initiale }
B104   BEGIN
B11C     I:=I+1;
B123     IF A[I]=' '
B140        THEN SPINIT
B14A        ELSE WHILE A[I]<>'%' DO { caracter <> % ? }
B17F             BEGIN
B17F               J:=J+1;       { memoreaza }
B186               B[J]:=A[I];  { caracter  }
B1BA               CUVINT       { trateaza cuvint }
B1BA             END
B1C3   END;
B1CC
B1CC   PROCEDURE SPATIU;
B1CF     { trateaza spatii inter-cuvinte }
B1CF   BEGIN
B1E7     I:=I+1;
```

```
B1EE    IF A[I]=' '
B20B        THEN SPATIU
B215        ELSE WHILE A[I]<>'%' DO
B24A               BEGIN
B24A                 J:=J+1;B[J]:=' '; { memoreaza un spatiu}
B26E                 J:=J+1;B[J]:=A[I];{ si primul caracter }
B2A9                 CUVINT             { trateaza cuvint }
B2A9               END
B2B2 END;
B2BB
B2BB PROCEDURE CUVINT;
B2BE   { memoreaza un cuvint }
B2BE BEGIN
B2D6   I:=I+1;
B2DD   IF A[I]=' '
B2FA     THEN SPATIU
B304     ELSE WHILE A[I]<>'%' DO
B339            BEGIN
B339              J:=J+1;
B340              B[J]:=A[I];
B374              CUVINT
B374            END
B37D END;
B386
B386 BEGIN
B38F   I:=0;J:=0;
B39B   { citire text }
B39B   REPEAT
B39B     READ(A[I]);
B3BC     I:=I+1
B3C4   UNTIL A[I-1]='%';
B3EB   { tratare text }
B3EB   I:=0;
B3F1   SPINIT;
B3F6   { scrie text tratat }
B3F6   I:=1;
B3FC   REPEAT
B3FC     WRITE(B[I]);
B41B     I:=I+1
B423   UNTIL I>J;
B435   WRITELN
B435 END {$P}.
```

P.V.39.

Probleme propuse

1. Să se scrie o procedură care să realizeze împărţirea a două numere întregi utilizînd doar operaţii de adunare şi scădere.

2. Să se verifice şi să se discute corectitudinea următoarelor declaraţii de proceduri şi apelurile lor:

a)
```
TYPE T=(BINE,RAU,POTRIVIT);
VAR A:INTEGER;
    B:T;
PROCEDURE P(A:T);
  BEGIN
    IF A=5
      THEN WRITELN('NU E BINE')
      ELSE WRITELN('E BINE')
  END;
............
BEGIN
............
  A:=7;
  P(RAU);
............
END.
```

b)
```
VAR X:INTEGER;
PROCEDURE P(VAR A:REAL;B:REAL);
  BEGIN
    A:=B+5
  END;
............
BEGIN
  ...........
  P(X+7,X);
  ...........
END.
```

3. Să se realizeze un subprogram care să calculeze valoarea 2^{300}. Să se transforme apoi subprogramul pentru calculul oricărei puteri de forma N^M, N<10, M întreg pozitiv.

4. Să se realizeze o procedură PAUSE(N:INTEGER) care să aibă acelaşi efect ca şi instrucţiunea PAUSE din BASIC.

5. Să se realizeze o procedură care roteşte cu 90^o o matrice pătratică.

6. Să se realizeze o procedură pentru formarea unei linii text dintr-o pagină astfel: procedura primeşte un cuvînt pe care îl adaugă la linia în curs de formare (cuvintele sînt separate printr-un spaţiu); dacă prin adăugarea cuvîntului lungimea liniei ar depăşi 64 caractere, acesta urmează să fie primul cuvînt din linia următoare.

7. Aceeaşi problemă, dar tipărirea liniei să se facă după "aranjarea" (filarea) ei, prin adăugarea de spaţii între cuvinte, dacă este cazul, astfel încît ultimul cuvînt al liniei să fie aliniat la dreapta (ultimul caracter al liniei este al 64-lea).

8. Să se realizeze procedurile de bază ale unui editor de texte care lucrează asupra unei pagini de text formată din 32 de rînduri, fiecare rînd avînd 64 caractere:
-localizarea unui şir de caractere: LOC(X:SIR);
-inserarea unui şir de caractere: INS(X:SIR);
-ştergerea unui şir: STG(X:SIR);
-înlocuirea unui şir cu alt şir: INL(X,Y:SIR);
-poziţionarea la începutul unei linii: POZ(N:INTEGER).

9. Să se realizeze un program care să analizeze o pagină de text de 50 de rînduri, fiecare rînd avînd 64 caractere; programul va afişa toate cuvintele o singură dată şi rîndurile pe care apar cuvintele respective.

10. Presupunînd că avem în memorie două seturi de caractere astfel reprezentate încît pe un caracter obişnuit Spectrum încap două asemenea caractere -"stînga" şi "dreapta"-, să se realizeze un subprogram care să permită scrierea unui text cu astfel de caractere mici, deci pe un rînd al monitorului să poată fi scrise 64 de caractere.

11.a). Să se scrie subprograme care realizează diferite operaţii asupra valorilor complexe: adunare, scădere, înmulţire, împărţire, modul, argument, citire, scriere.
b). Să se folosească procedurile realizate la punctul a) pentru a realiza un program care să citească un şir de numere

complexe, să calculeze produsul lor P şi suma S precum şi expresia:

$$E = \left(\frac{P - S}{P*S} \right)^n$$

Calculul se va face în două moduri: direct şi folosind formula lui Moivre. Se vor tipări rezultatele obţinute prin ambele metode.

12. Să se modifice programul **LABIRINT** (P.V.37), adăugîndu-i-se o procedură pentru generarea chiar în program a unui labirint.

13. Să se realizeze un subprogram de tip funcţie pentru ridicarea unui număr întreg X la o putere naturală N.

14. Să se realizeze o funcţie PUTERE pentru calculul expresiei X^Y, unde X şi Y sînt de tip REAL.

15. Se cere o funcţie POINT, care să realizeze aceeaşi operaţie ca aceea din BASIC.

16. Să se realizeze o funcţie CIFRA(N,M) care returnează valoarea celei de a M-a cifre, de la dreapta spre stînga, a numărului N, scris în sistemul zecimal.
Exemplu: CIFRA(7283,3)=2.

17. Să se realizeze funcţia recursivă care calculează valorile funcţiei lui Ackermann definită pentru m>=0 şi n>=0 astfel:

Ack(0,n)=n+1
Ack(m,0)=Ack(m-1,1) pentru m>0
Ack(m,n)=Ack(m-1,Ack(m,n-1)) pentru m>0 şi n>0.

18. Să se realizeze funcţia recursivă pentru calculul valorilor polinoamelor lui Hermite, H(x), ştiind că:

$H_0(x)=1$

$H_1(x)=2x$

$H_n(x)=2H_{n-1}(x)-2(n-1)H_{n-2}(x)$ pentru n>1

Să se realizeze şi o funcţie iterativă şi să se compare cele două metode de calcul.

19. Dîndu-se un şir de funcţii şi variabile sub forma unui şir de caractere şi aritatea fiecărei funcţii, să se modifice şirul iniţial, folosind apelul recursiv, punîndu-se la locul lor parantezele şi virgulele.

Exemplu:
-şirul iniţial: FXGXYHZ
-arităţile: ar(F)=3
ar(G)=2
ar(H)=1
- se va obţine: F(X,G(X,Y),H(Z)).
(Prin aritatea unei funcţii înţelegem numărul argumentelor acelei funcţii).

VI. ALOCAREA DINAMICA A MEMORIEI

In limbajul PASCAL se poate lucra cu
-*structuri de date statice* şi cu
-*structuri dinamice*.
Structurilor statice
-li se alocă spaţiu pe parcursul *compilării* şi
-pot fi referite prin *nume*.
Structurile dinamice
-*nu vor avea nume* şi
-nu li se alocă spaţiu decît în timpul *execuţiei*
-în plus, acest spaţiu se poate elibera la cererea explicită a programatorului şi astfel, se poate reda sistemului de gestiune a memoriei.

6.1. TIPUL REFERINTA (POINTER)

Structurile dinamice, neavînd nume, sînt accesibile prin intermediul altor variabile, care conţin adresele lor, şi în felul acesta "arată spre ele", "le indică". Aceste variabile se numesc *pointeri* şi formează *tipul referinţă* (tipul pointer sau tipul reper). *Variabilele de tip pointer sînt variabile statice obişnuite şi au ca valori adrese*, care pot fi adresele de început ale unor structuri dinamice sau a unei componente a unei asemenea structuri. Prin urmare accesul la o variabilă dinamică, indicată de o variabilă pointer se realizează prin *adresarea indirectă* a variabilei pointer. Această indirectare din punct de vedere formal se pune în evidenţă prin ataşarea caracterului '^' după numele variabilei pointer.

Variabila dinamică, indicată de o variabilă pointer, se va referi, nu printr-un nume propriu, ci prin intermediul pointerului său: *variabilă de tip pointer^*.

Exemplu:
P - variabilă pointer,
P^ - variabilă dinamică indicată de pointerul P.

In PASCAL există o legătură fixă între o variabilă pointer şi tipul variabilelor indicate. O variabilă pointer dată se poate referi la variabile indicate de un anumit tip, conform definirii sale.
Pointerul nul se identifică prin cuvîntul rezervat *NIL*; el este compatibil cu orice tip pointer.

Un *tip pointer* se defineşte precizînd tipul variabilei indicate, precedat de caracterul '^'

Exemple:
a) TYPE TIPPOINTER=^T;

```
b) TYPE COMPLEX=RECORD
                  REEL,IMAG:REAL
                END;
   VAR P:^COMPLEX;
```

Variabila dinamică din exemplul anterior poate fi vizualizată ca în fig 6.1.

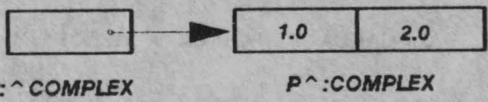


fig. 6.1

Se observă că simbolul '^' poate precede un identificator de tip, definind astfel un tip pointer, sau poate urma un identificator de variabilă pointer, precizînd în acest caz o variabilă indicată.

Menţionăm că în limbajul *PASCAL standard* şi în majoritatea implementărilor sale este permisă şi o secvenţă de declaraţii de forma:

```
TYPE TIPPOINTER=^T;
         T=...
```

adică *o referire a identificatorului T înainte ca el să fie definit*.

Alocarea memoriei pentru o structură dinamică se realizează cu ajutorul **procedurii NEW** care are un singur parametru de tip pointer.

Exemplu:

Fie P o variabilă de tip pointer. Efectul apelului de procedură NEW(P) este de *a rezerva o zonă de memorie pentru o variabilă indicată P^ şi de a iniţializa variabila pointer P cu adresa zonei rezervate.* Menţionăm că *NEW(P) nu realizează şi iniţializarea variabilei dinamice indicate de P.*

Pentru exemplul anterior, efectul instrucţiunii NEW(P) este de a rezerva o zonă de memorie necesară pentru o variabilă de tip COMPLEX, (fără a înregistra valori în această zonă) şi de a iniţializa variabila P cu adresa zonei rezervate. Prin urmare, această zonă se poate indica (referi) prin P^.

Observaţii:

1)-mai multe variabile pointer pot referi aceeaşi variabilă indicată conform fig.6.2.

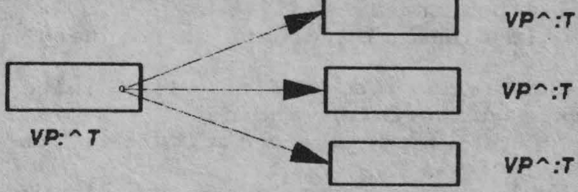


fig.6.2

2)-mai multe variabile indicate pot fi referite de aceeaşi variabilă pointer conform fig.6.3). Situaţia din figură se poate realiza în timp; un pointer indică la un moment dat o singură variabilă dinamică, dar schimbîndu-şi valoarea, îşi va schimba şi indicaţia.

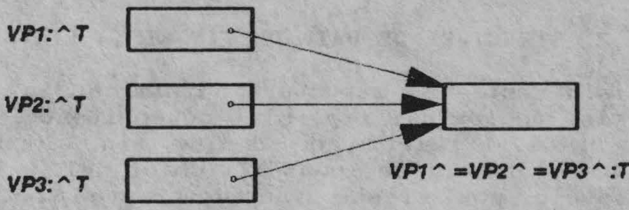


fig.6.3

Programul **POINT1** (P.VI.1) exemplifică (în liniile 150-200) modul în care se poate introduce şi afişa un număr complex cu structură articol (RECORD). In liniile 210-260 se generează acelaşi număr complex ca variabilă dinamică, iar în liniile 290-350 se generează zece numere complexe prin intermediul aceleiaşi variabile de tip pointer.

```
B000   20 {INTRODUCERE,AFISARE NR.COMPLEXE,GENERARE,AFISARE NR.COMPLEXE}
B000   30 PROGRAM POINT1;
B000   40 CONST P=CHR(16);
B000   50 TYPE COMPLEX=RECORD
B000   60                RE,IM:REAL
B000   70                END;
B000   80 VAR Z:COMPLEX;
B009   90     PZ:^COMPLEX;
B009  100     C:CHAR;
B009  110     N,I:INTEGER;
B009  120 BEGIN
B012  130  PAGE;
B017  140  WRITELN('DATI NR.COMPLEX:');
B035  150  READ(Z.RE,Z.IM);
B057  160  IF Z.IM<0 THEN
B079  170               C:='-'
B07C  180             ELSE
B081  190               C:='+';
B086  200  WRITELN('Z=',Z.RE:5:2,C,ABS(Z.IM):5:2,'I');
B0D3  210  NEW(PZ);
B0DC  220  PZ^.RE:=Z.RE;
B0F5  230  PZ^.IM:=Z.IM;
B112  240  WRITELN(P);
B11A  250  WRITELN('PRIN POINTER:');
B135  260  WRITELN('Z=',PZ^.RE:5:2,C,ABS(PZ^.IM):5:2,'I');
B186  270 { O SINGURA VARIABILA POINTER}
B186  280 { GENEREAZA MAI MULTE VARIABILE INDICATE}
B186  290  FOR I:=1 TO 10 DO
B1A0  300   BEGIN
B1A3  310       NEW(PZ);
B1AC  320       PZ^.RE:=I;
B1C0  330       PZ^.IM:=2*I;
B1E3  340       WRITELN('Z',I,'=',PZ^.RE:2:0,'+',PZ^.IM:2:0,'I')
B234  350   END;
B23A  360  WRITELN(P);
B242  370 END {$P}.
```

```
PRIN POINTER:
Z= 5.65- 7.83I
Z1 = 1+ 2I
Z2 = 2+ 4I
Z3 = 3+ 6I
Z4 = 4+ 8I
Z5 = 5+10I
Z6 = 6+12I
Z7 = 7+14I
Z8 = 8+16I
Z9 = 9+18I
Z10 =10+20I
```

P.VI.1

6.2. STRUCTURI DE DATE DE TIP LISTA LINIARA

Lista este o structură dinamică în care toate elementele sînt de *acelaşi tip*, sînt concomitent în memorie şi între ele există o relaţie de ordine. In timpul execuţiei numărul elementelor variază, poate fi chiar nul.

Listele pot creşte sau descreşte (pot fi create începînd cu primul element pînă la ultimul sau invers, de la ultimul spre primul) elementele lor putînd fi referite, inserate sau şterse în/din orice poziţie din cadrul listei.

Listele pot fi concatenate sau scindate în subliste.

Lista se mai poate defini ca fiind o secvenţă de mai multe elemente (noduri) de un anumit tip, numit tip de bază(T_B)

Pentru n=0 lista este vidă.

$a_1, a_2, \ldots, a_n, n>=0, a_i:T_B$

Pentru n>=1, a_1= primul nod al listei;

a_n= ultimul nod al listei.

Proprietăţi:

-nodurile pot fi ordonate liniar în funcţie de poziţia lor în cadrul listei:

-a_i precede pe a_{i+1}, i=1,2,...,n-1;

-a_i succede pe a_{i-1}, i=2,3,...,n;

-nodul i se află pe poziţia a i-a.

Observaţie:

Lista liniară în principiu este o structură dinamică de date, dar ea nu poate fi implementată dinamic decît în limbajele de programare care posedă mecanismul de alocare dinamică.

6.2.1. Implementarea listelor cu ajutorul tipului pointer (liste înlănţuite)

Utilitatea tipului pointer devine evidentă în cazul structurilor de date dinamice *înlănţuite*, structuri care în general se definesc recursiv.

Inlănţuirea se pune în evidenţă cu ajutorul unor *informaţii de legătură* care la rîndul lor pot fi memorate în unul sau mai multe cîmpuri ale articolului care conţine informaţiile ataşate unui nod (element) al listei. Evident, *informaţiile propriu-zise* ataşate nodurilor pot fi memorate de asemenea în unul sau mai multe cîmpuri.

informaţii propriu-zise	informaţii de legătură

fig.6.4

O listă liniară poate fi definită în termeni recursivi *în limbajul PASCAL standard* astfel:

```
TYPE LISTA=^NOD;
     NOD=RECORD
           CHEIE:INTEGER;
            INFO:...;
             URM:LISTA
         END;
```

Se observă că identificatorul NOD este referit înainte de a fi definit. Deoarece acest lucru este nepermis în HP4TM, definirea listei liniare în această implementare se face în felul următor:

```
TYPE NOD=RECORD
           CHEIE:INTEGER;
            INFO:...;
             URM:^NOD;
         END;
     LISTA=^NOD;
```

Orice nod din lista definită astfel are trei cîmpuri:
- o cheie pentru identificarea nodului (CHEIE) care nu este obligatorie;
- o zonă pentru informaţia utilă (INFO) care poate diferi ca structură şi tip în funcţie de problemă;
- un pointer de înlănţuire la nodul următor (URM).

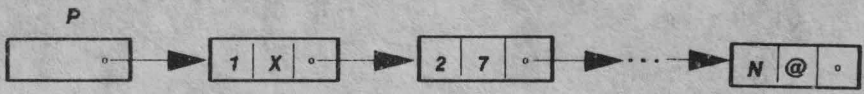


fig.6.5

6.2.2. Tehnici de inserare a nodurilor

a. Inserare la începutul listei

Vom utiliza următoarele notaţii:
P-variabilă pointer care indică primul nod al listei, necesară deoarece *fiecare* nod al listei se inserează la începutul ei;
Q-variabilă de lucru de tip pointer.

Secvenţa de inserare este următoarea:

```
NEW(Q);
Q^.URM:=P;
P:=Q;
```

care trebuie completată cu iniţializarea cîmpurilor informaţiei propriu-zise.

Efectul primei instrucţiuni este alocarea spaţiului necesar cîmpurilor noului nod şi iniţializarea pointerului Q cu adresa acestui nod. In urma celei de-a doua instrucţiuni se realizează legătura dintre noul nod şi primul nod al listei (cîmpul de tip pointer al noului nod va indica vechiul nod din listă). Operaţiile descrise sînt vizualizate în fig.6.6.

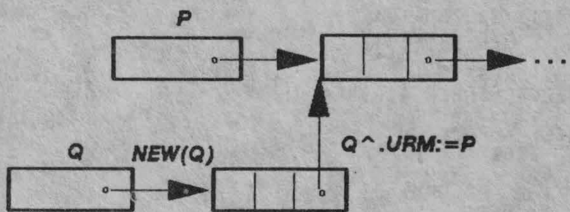


fig.6.6

In urma celei de-a treia instrucţiuni se modifică conţinutul pointerului P astfel încît el să indice nodul inserat (fig.6.7).

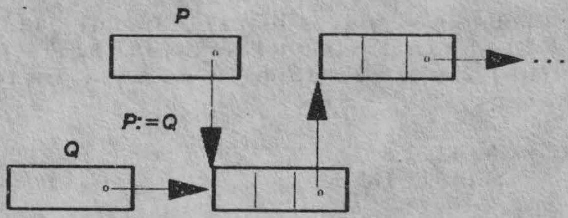


fig. 6.7

Avînd în vedere că o listă se construieşte inserînd elementele în lista iniţial vidă, secvenţa de program PASCAL pentru construirea unei liste este următoarea:

```
P:=NIL;
WHILE N>0 DO      {N REPREZINTA NR. ELEMENTELOR DIN LISTA}
  BEGIN
    NEW(Q);
    Q^.URM:=P;
    P:=Q;
    Q^.CHEIE:=N;
    N:=N-1
  END;
```

Observaţie:

In secvenţa de mai sus, instrucţiunea WHILE se poate înlocui cu FOR, dar deoarece nu întotdeauna se cunoaşte numărul elementelor (întrucît ele se selectează pe baza anumitor criterii), instrucţiunea WHILE este mai adecvată.

Programul **LISTAPOINT** (P.VI.2) creează o listă prin tehnica inserării la începutul listei şi o afişează, traversînd-o (traversarea se va trata în subparagraful 6.2.4).

```
B0C4  { INSERTIE NODURI LA INCEPUTUL UNEI LISTE SI AFISARE }
B0C4  PROGRAM LISTAPOINT1;
B0C4  TYPE TAB=ARRAY[1..20]OF CHAR;
B0C4       NOD=RECORD
B0C4            CHEIE:INTEGER;
B0C4            NUME:TAB;
B0C4            URM:^NOD
B0C4           END;
B0C4  VAR P,Q:^NOD;
B0CD      N,I:INTEGER;
B0CD      NUM:TAB;
B0CD  PROCEDURE CITNUM;
B0D0  BEGIN
B0E8    WRITE('DATI NUMELE:');
B0FF    READLN;READ(NUM);
B10A  END; { CITNUME }
B110  PROCEDURE PREGTIP;
B113  BEGIN
B12B    FOR I:=20 DOWNTO 1 DO
B144        IF NUM[I]=CHR(0)
B165         THEN NUM[I]:=CHR(32)
B18D  END; { PREGTIP }
B198  PROCEDURE INSEREAZAINFATA;
B19B  BEGIN
B1B3    WHILE N>0 DO
B1C9         BEGIN
B1C9            NEW(Q);
B1D2            Q^.URM:=P;
B1E2            P:=Q;
B1E8            Q^.CHEIE:=N;
B1F4            CITNUM;
B1FD            PREGTIP;
B206            Q^.NUME:=NUM;
B215            N:=N-1
B21D         END
B21C  END; { INSEREAZAINFATA }
```

```
B225 PROCEDURE TRAVLISTA;
B228 BEGIN
B240   Q:=P;
B246   WHILE Q<>NIL DO
B252       BEGIN
B25B           WRITELN;
B25E           WITH Q^ DO
B269               WRITELN(CHEIE,'  ',NUME);
B290           Q:=Q^.URM
B298       END
B29E END; { TRAVLISTA }
B2A7 BEGIN { MAIN }
B2B0  WRITE('DATI NR.DE NODURI:');
B2CD  READ(N);
B2D3  P:=NIL;
B2D9  INSEREAZAINFATA;
B2DE  WRITELN(CHR(16));
B2E8  WRITELN;
B2EB  WRITE('LISTA A FOST CREATA SI CONTINE NODURILE:');
B31E  TRAVLISTA;
B323  WRITE(CHR(16))
B32A END {$P}.
```

```
LISTA A FOST CREATA SI CONTINE NODURILE:
1   PACURARU MIHAI

2   SPERNEAC PAUL

3   RADULESCU CORNELIU

4   HERBEI CARINA

5   SERBAN MIHAELA

6   TUDOR MIRON
```

P.VI.2

b. Inserare la sfîrşitul listei

Vom utiliza următoarele notaţii:

P-variabilă pointer care indică primul nod, necesară pentru traversarea listei;

Q-variabilă pointer care indică ultimul nod al listei, necesară deoarece fiecare nod se inserează la sfîrşitul ei

R-variabilă de lucru de tip pointer.

Inserarea unui element la sfîrşitul listei se realizează cu următoarea secvenţă de program PASCAL:

```
NEW(R);
Q^.URM:=R;
R^.URM:=NIL;
Q:=R;
```

Inainte de inserare lista are forma din fig.6.8.

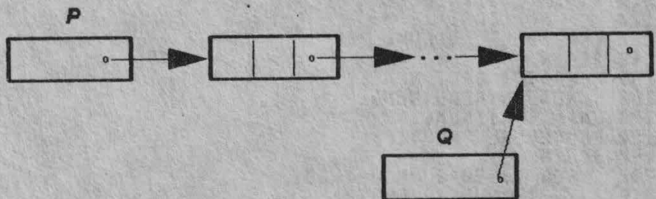


fig.6.8

In urma execuţiei primelor trei instrucţiuni din secvenţa anterioară, lista are configuraţia din fig.6.9.

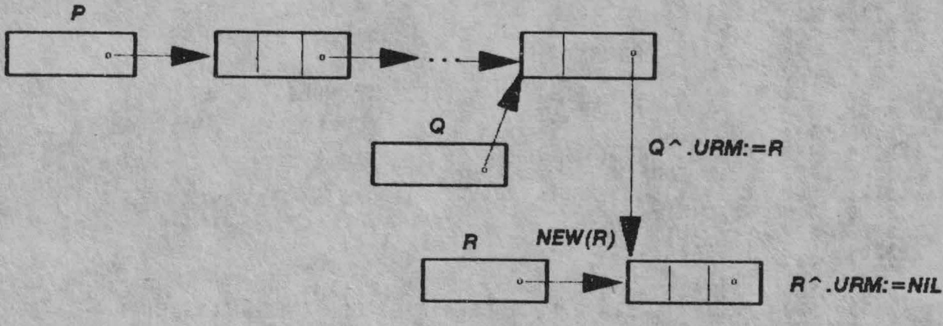


fig.6.9

Pentru a putea introduce secvenţa de inserare într-o structură repetitivă, *pointerul Q trebuie să indice mereu ultimul element din listă, deci variabila Q trebuie actualizată* (instrucţiunea a patra). In final, lista va avea structura din fig.6.10.

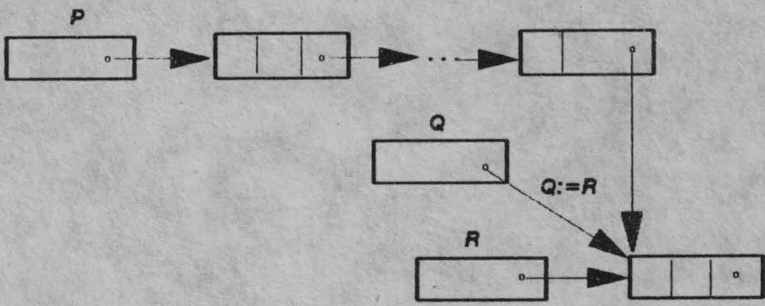


fig.6.10

Observaţie:

Deoarece într-o listă vidă Q^.URM nu există, primul nod trebuie inserat într-un alt mod, prezentat în programul P.VI.3.

Programul **LISTAPOINT2** (P.VI.3) creează o listă prin tehnica inserării la sfîrşitul listei.

```
B170   20 { INSERTIE NODURI IN SPATELE UNEI LISTE SI AFISARE }
B170   30 PROGRAM LISTAPOINT2;
B170   40 TYPE TAB=ARRAY[1..20]OF CHAR;
B170   50      NOD=RECORD
B170   60           CHEIE:INTEGER;
B170   70           NUME:TAB;
B170   80           URM:^NOD
B170   90         END;
B170  100 VAR P,Q,R:^NOD;
B179  110     N,I,J:INTEGER;
B179  120     NUM:TAB;
B179  130 PROCEDURE CITNUM;
B17C  140 BEGIN
B194  150   WRITE('DATI NUMELE:');
B1AB  160   READLN;READ(NUM);
B1B6  170 END; { CITNUME }
B1BC  180 PROCEDURE PREGTIP;
B1BF  190 BEGIN
B1D7  200   FOR I:=20 DOWNTO 1 DO
B1F0  210      IF NUM[I]=CHR(0)
B211  220       THEN NUM[I]:=CHR(32)
B239  230 END; { PREGTIP }
```

```
B244 PROCEDURE TRAVLISTA;
B247 BEGIN
B25F   Q:=P;
B265   WHILE Q<>NIL DO
B27A       BEGIN
B27A           WRITELN;
B27D           WITH Q^ DO
B288               WRITELN(CHEIE,'   ',NUME);
B2AF           Q:=Q^.URM
B2B7       END
B2BD END; { TRAVLISTA }
B2C6 PROCEDURE INSEREAZAPRIMUL;
B2C9 BEGIN
B2E1  P:=NIL;
B2E7  NEW(Q);
B2F0  WRITE('DATI CHEIA:');
B306  READ(Q^.CHEIE);
B312  P:=Q;
B318  CITNUM;
B321  PREGTIP;
B32A  Q^.NUME:=NUM
B334 END; { INSEREAZAPRIMUL }
B33F PROCEDURE INSEREAZAREST;
B342 BEGIN
B35A  NEW(R);
B363  R^.URM:=NIL;
B373  Q^.URM:=R;
B383  WRITE('DATI CHEIA:');
B399  READ(R^.CHEIE);
B3A5  CITNUM;
B3AE  PREGTIP;
B3B7  R^.NUME:=NUM;
B3C6  Q:=R
B3C6 END; { INSEREAZAREST }
B3D2 BEGIN  { MAIN }
B3DB  WRITE('DATI NR.DE NODURI:');
B3F8  READ(N);
B3FE  INSEREAZAPRIMUL;
B403  FOR J:=2 TO N DO
B421   INSEREAZAREST;
B42D  WRITELN(CHR(16));
B437  WRITELN('LISTA ESTE URMATOAREA:');
B45B  TRAVLISTA;
B460  WRITELN(CHR(16))
B467  END {$P}.
```

```
LISTA ESTE URMATOAREA:

10   POP ION

20   IONICA FLORIN

25   COSTESCU MARIUS

40   GHITA IOSIF

66   DARIE IOANA
```

P.VI.3

c. Inserare într-un loc oarecare al listei

Notăm cu:

P-un pointer care indică nodul listei după care se doreşte inserarea;

Q-o variabilă de lucru de tip pointer.

Inserarea unui nod *după* nodul P^ se realizează cu următoarea secvenţă de program PASCAL:

```
NEW(Q);
Q^.URM:=P^.URM;
P^.URM:=Q;
```

Inainte de inserare, lista are structura din fig.6.11

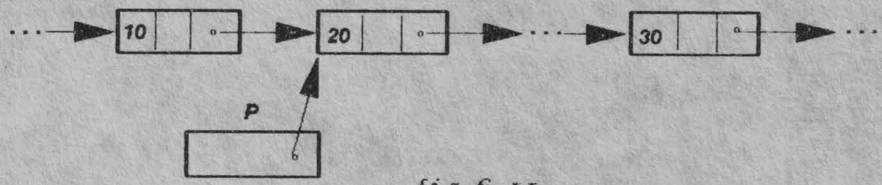


fig.6.11

In urma secvenței anterioare de instrucțiuni, lista va avea configurația din fig.6.12.

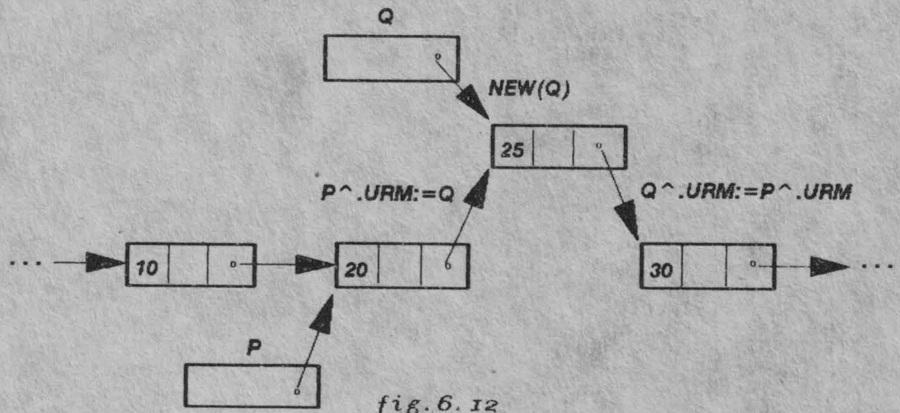


fig.6.12

Observație:

Dacă se dorește inserarea unui nod înaintea unui nod indicat P^, neavînd nici un pointer care să indice predecesorul lui P^, inserarea se va realiza prin următoarea tehnică:

- -se inserează un nod după P^;
- -se inițializează acest nod cu P^, creîndu-se astfel o "dublură" a variabilei indicate de pointerul P;
- -cîmpurile CHEIE și INFO din vechiul nod P^ se actualizează corespunzător noului nod.

Secvența de program PASCAL care realizează o inserare în acest fel este următoarea:

```
NEW(Q);
Q^:=P^;
P^.URM:=Q;
P^.CHEIE:=CHEIE;
P^.INFO:=INFO;
```

Efectul primelor două instrucțiuni se poate urmări în fig.6.13.

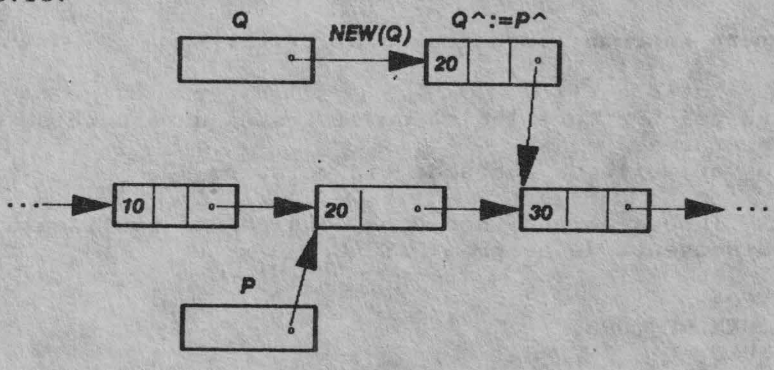


fig.6.13

După crearea acestei "dubluri" a lui P^ urmează legarea lui după vechiul P^, conform celei de-a treia instrucţiuni, aşa cum rezultă din fig.6.14.

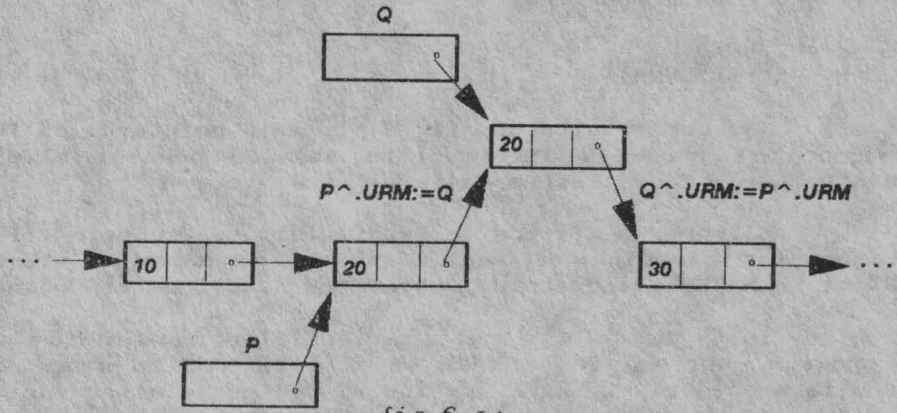


fig.6.14

Astfel, s-a pregătit loc pentru noul nod al listei în vechiul P^ ale cărui cîmpuri se vor iniţializa cu informaţiile nodului nou (fig.6.15).

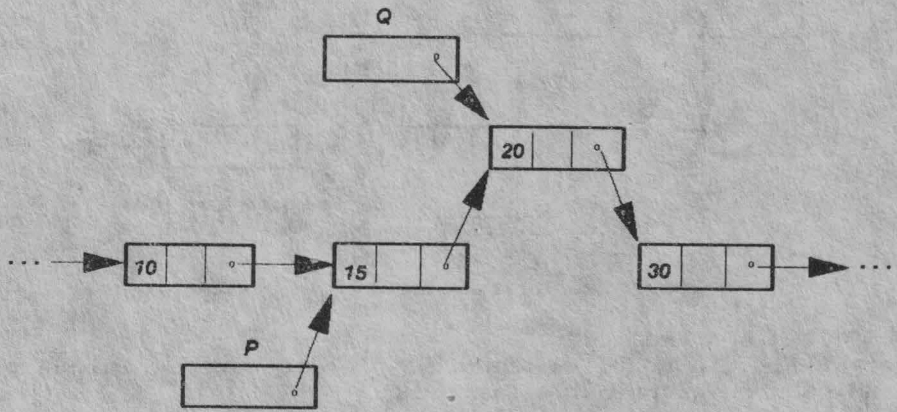


fig.6.15

6.2.3. Tehnici de suprimare (ştergere) a nodurilor unei liste

Considerăm o listă (fig.6.16); se pune problema ştergerii unui nod al acesteia. Problema va avea rezolvări diferite în funcţie de poziţia nodului relativ la un pointer cunoscut P.

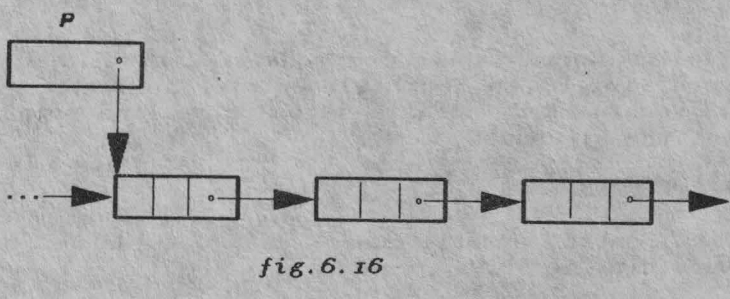


fig.6.16

1)-*dacă nodul pe care dorim să-l suprimăm este succesorul nodului P^*, şi în plus dorim să păstrăm posibilitatea de a ajunge la el, secvenţa instrucţiunilor care realizează suprimarea este următoarea:

```
R:=P^.URM;
P^.URM:=R^.URM;
```

Aşa cum rezultă din fig.6.17, prin pointerul de lucru R se poate ajunge la informaţiile ataşate nodului eliminat din structură, dacă acestea mai sînt necesare.

Eliberarea fizică a spaţiului de memorie alocat variabilelor dinamice se realizează cu **procedurile standard MARK şi RELEASE**, avînd acelaşi parametru actual de tip pointer.

Toată memoria alocată variabilelor dinamice în zona de program cuprinsă între MARK şi RELEASE se eliberează după execuţia procedurii RELEASE, putînd fi realocată altor variabile dinamice. Deoarece sînt distruse *toate variabilele dinamice create după executarea procedurii MARK*, procedurile MARK şi RELEASE se vor folosi cu precauţie, mai ales dacă se lucrează simultan cu mai multe structuri dinamice de date.

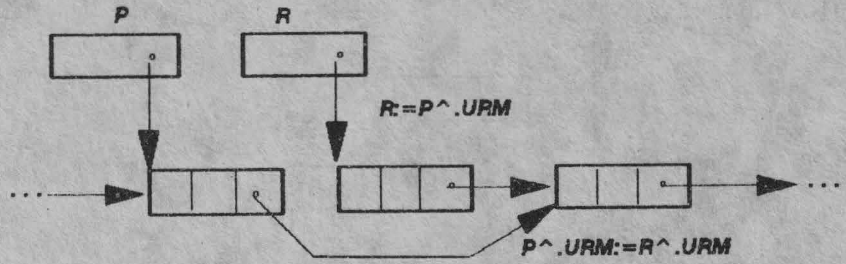


fig.6.17

Dacă ştergerea se realizează cu instrucţiunea:

```
P^.URM:=P^.URM^.URM
```

informaţiile din nodul eliminat se pierd deoarece nici un pointer nu indică nodul suprimat, (fig.6.18).

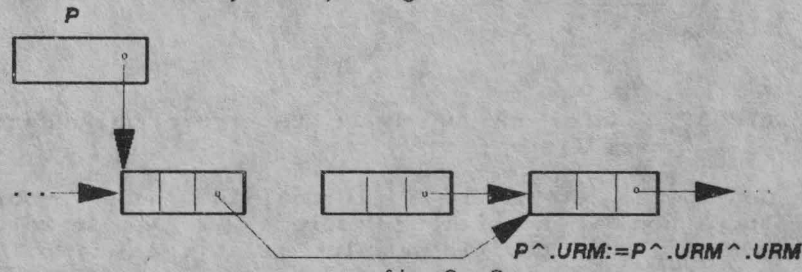


fig.6.18

2)-*dacă dorim suprimarea nodului indicat de P*, se aduce succesorul lui P^ în locul elementului indicat de P, apoi se suprimă vechiul nod; aceasta se poate realiza printr-o singură instrucţiune şi anume:

```
P^:=P^.URM^;
```

Pentru exemplificare, să considerăm o listă cu structura din fig.6.19.

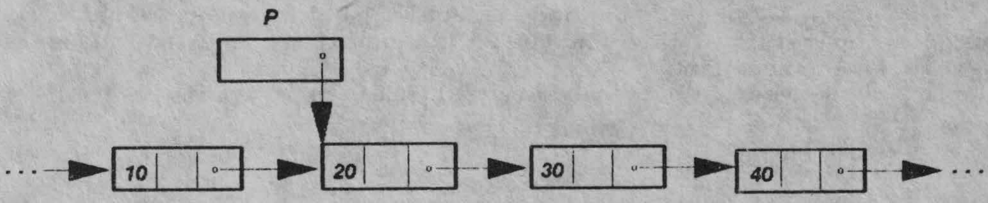


fig. 6.19

Să presupunem că dorim să ştergem elementul cu cheia 20. Efectul instrucţiunii de mai sus se poate urmări în fig. 6.20.

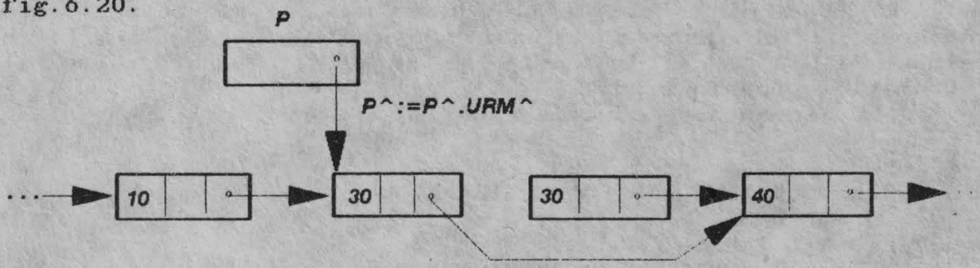


fig. 6.20

Dacă trebuie păstrată posibilitatea utilizării informaţiilor propriu-zise ataşate nodului suprimat, ştergerea se va realiza cu instrucţiunile:

```
R:=P^.URM;
P^:=R^;
```

Dacă lista are configuraţia din fig. 6.19, în urma execuţiei primelor două instrucţiuni din secvenţa de mai sus structura listei va fi cea din fig. 6.21.

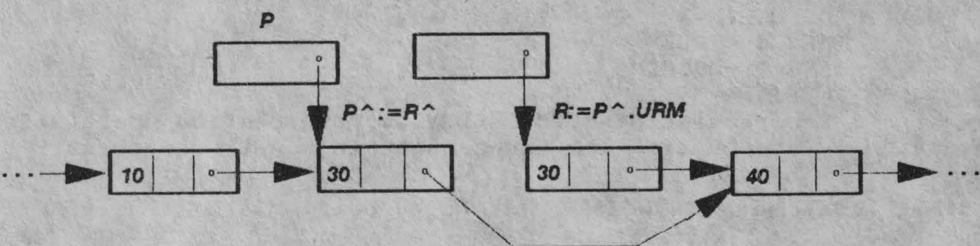


fig. 6.21

Observaţie:
Aceste tehnici se pot aplica numai dacă P^ nu este ultimul nod al listei, deoarece în acest caz nu există variabilă dinamică indicată de pointerul P^.URM, el avînd valoarea NIL.

6.2.4. Traversarea unei liste înlănţuite

A traversa o listă înlănţuită înseamnă a trece de la nod la nod în vederea executării anumitor operaţii asupra tuturor nodurilor listei.

Pentru traversare se vor folosi doi pointeri:
- -P indică primul nod al listei;
- -Q un pointer de lucru.

Dacă Q^ este un nod oarecare al listei, vom nota cu O(Q^) o operaţie a cărei natură în contextul de faţă nu este nevoie să o precizăm.

Secvenţa de traversare a listei este următoarea:

```
Q:=P;
WHILE Q<>NIL DO
  BEGIN
    {O(Q^);}
    Q:=Q^.URM
  END;
```

Una din operaţiile cele mai frecvente este *căutarea*, adică depistarea unui nod avînd cheia egală cu o valoare dată, X. (Prin cheie se înţelege oricare cîmp al informaţiei propriu-zise.)

Secvenţa de căutare este:

```
Q:=P;
WHILE (Q<>NIL) AND (Q^.CHEIE<>X) DO
  Q:=Q^.URM;
IF Q=NIL THEN
          WRITE('NOD INEXISTENT')
        ELSE
          WRITE('PRIMUL NOD CU CHEIA DORITA=',Q^.INFO);
```

Observaţie:

Dacă Q=NIL atunci nodul Q^ nu există, şi în acest caz o eventuală evaluare (dependentă de implementare) a subexpresiei (Q^.CHEIE<>X) ar genera o eroare. O variantă corectă independent de implementare este următoarea:

```
B:=FALSE;
Q:=P;
WHILE (Q<>NIL) AND NOT B DO
  IF Q^.CHEIE=X
   THEN B:=TRUE
   ELSE Q:=Q^.URM;
```

Programul **LISTAPOINT2** (P.VI.4) exemplifică operaţiile executabile asupra listelor, prezentate în acest subparagraf.

```
B8BE  { CREAZA PRIN INSERARE IN FATA SAU IN SPATE,AFISEAZA,CAUTA,SUPRI
MA }
B8BE  PROGRAM LISTAPOINT2;
B8BE  TYPE TAB=ARRAY[1..20]OF CHAR;
B8BE       NOD=RECORD
B8BE            CHEIE:INTEGER;
B8BE            NUME:TAB;
B8BE            URM:^NOD
B8BE           END;
B8BE  VAR P,Q,R:^NOD;
B8C7      N,M,I,J,K,X:INTEGER;
B8C7      NUM:TAB;
B8C7      B:BOOLEAN;
B8C7  PROCEDURE CITNUM;
B8CA  BEGIN
B8E2    WRITE('DATI NUMELE:');
B8F9    READLN;READ(NUM);
B904  END; { CITNUME }
B90A  PROCEDURE PREGTIP;
B90D  BEGIN
B925    FOR I:=20 DOWNTO 1 DO
B93E       IF NUM[I]=CHR(0)
B95F        THEN NUM[I]:=CHR(32)
B987  END; { PREGTIP }
```

```
B992  PROCEDURE TRAVLISTA;
B995  BEGIN
B9AD   Q:=P;
B9B3   WHILE Q<>NIL DO
B9C8        BEGIN
B9C8             WRITELN;
B9CB             WITH Q^ DO
B9D6                  WRITELN(CHEIE,'   ',NUME);
B9FD             Q:=Q^.URM
BA05        END
BA0B  END; { TRAVLISTA }
BA14  PROCEDURE INSEREAZAPRIMUL;
BA17  BEGIN
BA2F   P:=NIL;
BA35   NEW(Q);
BA3E   WRITE('DATI CHEIA:');
BA54   READ(Q^.CHEIE);
BA60   P:=Q;
BA66   CITNUM;
BA6F   PREGTIP;
BA78   Q^.NUME:=NUM
BA82  END; { INSEREAZAPRIMUL }
BA8D  PROCEDURE INSEREAZAREST;
BA90  BEGIN
BAA8   NEW(R);
BAB1   R^.URM:=NIL;
BAC1   Q^.URM:=R;
BAD1   WRITE('DATI CHEIA:');
BAE7   READ(R^.CHEIE);
BAF3   CITNUM;
BAFC   PREGTIP;
BB05   R^.NUME:=NUM;
BB14   Q:=R
BB14  END; { INSEREAZAREST }
BB20  PROCEDURE MENIU;
BB23  BEGIN
BB3B   WRITELN('1. CREARE LISTA');
BB58   WRITELN('2. AFISARE LISTA');
BB76   WRITELN('3. INSERARE DUPA UN NOD');
BB9B   WRITELN('4. INSERARE INAINTEA UNUI NOD');
BBC6   WRITELN('5. CAUTARE NOD DUPA CHEIE');
BBED   WRITELN('6. SUPRIMARE NOD');
BC0E   WRITELN('7. STOP');
BC20  END; { MENIU }
BC26  PROCEDURE CREARE;
BC29  BEGIN
BC41   WRITE('DATI NR.DE NODURI:');
BC5E   READ(N);
BC64   INSEREAZAPRIMUL;
BC6D   IF N>1 THEN
BC80              INSEREAZAREST;
BC89  END; { CREARE }
BC8F  PROCEDURE STAI;
BC92  BEGIN
BCAA   REPEAT UNTIL INCH<>CHR(0)
BCB5  END; { STAI }
BCC5  PROCEDURE CAUTA(VAR B:BOOLEAN);
BCC8  BEGIN
BCE0   B:=FALSE;
BCEA   Q:=P;
BCF0   WHILE (Q<>NIL) AND NOT B DO
BD11    IF Q^.CHEIE=X
BD19        THEN B:=TRUE
BD31        ELSE Q:=Q^.URM
BD3D  END; { CAUTA }
BD4D  PROCEDURE AFISEAZA;
BD50  BEGIN
BD68   PAGE;
BD6D   WRITELN;
BD70   WRITELN('LISTA ESTE URMATOAREA:');
BD94   TRAVLISTA
BD94  END; { AFISEAZA }
BDA3  PROCEDURE INSDNOD;
BDA6  BEGIN
BDBE   WRITE('DATI CHEIA DUPA CARE SE INSEREAZA:');
BDEB   READ(X);
BDF1   CAUTA(B);
BDFE   IF NOT B
BDFE    THEN
BE07        WRITE('CHEIA NU EXISTA!!!')
```

```
BE24    ELSE
BE27         BEGIN
BE27              WRITE('DATI CHEIA:');
BE3D              READ(M);
BE43              CITNUM;
BE4C              PREGTIP;
BE55              N:=N+1;
BE5C              NEW(R);
BE65              R^.CHEIE:=M;
BE71              R^.NUME:=NUM;
BE80              R^.URM:=Q^.URM;
BE98              Q^.URM:=R
BEA0         END
BEA8 END; { INSDNOD }
BEAE PROCEDURE INSINOD;
BEB1 BEGIN
BEC9  WRITE('DATI CHEIA INAINTEA CAREIA SE INSEREAZA:');
BEFC  READ(X);
BF02  CAUTA(B);
BF0F  IF NOT B
BF0F   THEN
BF18         WRITE('CHEIA NU EXISTA!!!')
BF35    ELSE
BF38         BEGIN
BF38              WRITE('DATI CHEIA:');
BF4E              READ(M);
BF54              CITNUM;
BF5D              PREGTIP;
BF66              N:=N+1;
BF6D              NEW(R);
BF76              R^:=Q^;
BF83              Q^.URM:=R;
BF93              Q^.CHEIE:=M;
BF9F              Q^.NUME:=NUM
BFA9         END
BFAE END; { INSINOD }
BFB4 PROCEDURE CAUTARE;
BFB7 BEGIN
BFCF  WRITE('DATI CHEIA DORITA:');
BFEC  READ(X);
BFF2  CAUTA(B);
BFFF  IF NOT B
BFFF   THEN
C008         WRITELN('CHEIA NU EXISTA!!!')
C025    ELSE
C02B         WRITELN('NODUL ESTE: ',Q^.CHEIE,' ',Q^.NUME);
C061 END; { CAUTARE }
C067 PROCEDURE SUPRIMARE;
C06A BEGIN
C082  WRITE('DATI CHEIA NODULUI CE SE VA SUPRIMA:');
C0B1  READ(X);
C0B7  CAUTA(B);
C0C4  IF NOT B
C0C4   THEN
C0CD         WRITE('CHEIA NU EXISTA!!!')
C0EA    ELSE
C0ED         R:=Q^.URM;
C0FB         Q^:=R^;
C108 END; { SUPRIMA }
C10E
C10E BEGIN  { MAIN }
C117  REPEAT
C117   STAI;
C11F   PAGE;
C124   WRITELN(CHR(18),CHR(1),CHR(22),CHR(5),CHR(14),'OPTIUNI',CHR(18
),CHR(0));
C16A   MENIU;
C16F   WRITE(CHR(22),CHR(21),CHR(10),'ALEGETI OPTIUNEA:');
C1A0   READ(K);
C1A6   CASE K OF
C1A9        1:CREARE;
C1BB        2:AFISEAZA;
C1CD        3:BEGIN INSDNOD;STAI END;
C1E4        4:INSINOD;
C1F6        5:CAUTARE;
C208        6:SUPRIMARE;
C21A        7:HALT
C224   END {CASE}
C227  UNTIL K=7
C22F END {$P}.
```

6.2.5. Aplicaţii ale listelor înlănţuite

a. Determinarea frecvenţei cuvintelor într-un text

Fiind dat un text format dintr-o succesiune de cuvinte, se cere să se afişeze cuvintele distincte şi numărul apariţiilor fiecărui cuvînt. Pentru rezolvare, se parcurge textul şi se depistează cuvintele. La prima apariţie cuvîntul se inserează în lista cuvintelor distincte, în rest se incrementează contorul care înregistrează numărul de apariţii ale cuvîntului. In final, traversînd lista, se vor afişa toate cuvintele distincte ale textului împreună cu numărul de apariţii ale fiecărui cuvînt.

Programul **CUVINTE** (P.VI.5) rezolvă această problemă pentru prima strofă din "Luceafărul" de Mihai Eminescu.

```
B1C0   { AFISEAZA CUVINTELE UNUI TEXT SI FRECVENTA LOR }
B1C0   PROGRAM CUVINTE;
B1C0   TYPE TAB=ARRAY[1..10]OF CHAR;
B1C0        NOD=RECORD
B1C0             CHEIE:TAB;
B1C0             NUMAR:INTEGER;
B1C0             URM:^NOD
B1C0            END;
B1C0        POINT=^NOD;
B1C0   VAR CUV:TAB;
B1C9       INCEPUT:POINT;
B1C9   PROCEDURE CAUTA(X:TAB);
B1CC   VAR Q:POINT;
B1CC       B:BOOLEAN;
B1CC   BEGIN
B1E4    Q:=INCEPUT;
B1ED    B:=TRUE;
B1F2    WHILE (Q<>NIL) AND B DO
B210     IF Q^.CHEIE=X
B21F        THEN
B231            B:=FALSE
B233        ELSE
B238            Q:=Q^.URM;
B24F     IF B
B24F        THEN
B256            BEGIN
B256             Q:=INCEPUT;
B25F             NEW(INCEPUT);
B268             WITH INCEPUT^ DO
B273              BEGIN
B273                CHEIE:=X;
B284                NUMAR:=1;
B294                URM:=Q
B29C              END
B2A7            END
B2A7        ELSE
B2AA            Q^.NUMAR:=Q^.NUMAR+1
B2C8   END; { CAUTA }
B2D5   PROCEDURE AFISARE(Q:POINT);
B2D8   VAR R:POINT;
B2D8   BEGIN
B2F0    R:=Q;
B2FC    PAGE;
B301    WRITE(CHR(22),CHR(1),CHR(0),'CUVINTE',' ':14,'FRECVENTA');
B34B    WRITELN;
B34E    WRITELN;
B351    WHILE R<>NIL DO
B369     BEGIN
B369        WRITELN(R^.CHEIE,' ':14,R^.NUMAR);
B39A        R:=R^.URM
B3A5     END
B3AE   END; { AFISARE }
B3B9   PROCEDURE PREGTIP;
B3BC   VAR I:INTEGER;
```

```
B3BC   BEGIN
B3D4    FOR I:=10 DOWNTO 1 DO
B3F6     IF CUV[I]=CHR(0)
B41A        THEN CUV[I]:=CHR(32)
B445   END; { PREGTIP }
B451   BEGIN   { MAIN }
B45A    INCEPUT:=NIL;
B460    PAGE;
B465    WRITELN('DATI CUVINTELE TEXTULUI SEPARATE PRIN <ENTER>(SFIRSIT=
<ENTER>)');
B4B1    READLN;
B4B4    READ(CUV);
B4BC    WHILE CUV[1]<>CHR(0) DO
B4E7     BEGIN
B4E7        PREGTIP;
B4EC        CAUTA(CUV);
B502        READLN;
B505        READ(CUV)
B50D     END;
B510    WRITELN(CHR(16));
B51A    AFISARE(INCEPUT);
B523    WRITELN(CHR(16));
B52D   END {$P}.
```

```
CUVINTE                 FRECVENTA

FATA                    1
FRUMOASA                1
PREA                    1
O                       1
IMPARATEST              1
MARI                    1
RUDE                    1
DIN                     1
NICIODATA               1
CA                      1
POVESTI                 1
CA-N                    1
ODATA                   1
FOST                    2
A                       2
```

P.VI.5

Observaţie:

Deoarece fiecare cuvînt nou s-a adăugat în listă la începutul ei, în urma traversării, cuvintele distincte apar în ordinea inversă citirii.

b. Metoda fanionului

Metoda aplicată în programul **CUVINTE** (P.VI.5) se poate perfecţiona prin *metoda fanionului* implementată în programul **FANION** (P.VI.6), metodă care constă în adăugarea unui nod fictiv în listă, referit de pointerul FANION care permite folosirea aceleiaşi tehnici de inserare şi pentru primul nod din listă.

```
B1F8   { AFISEAZA CUVINTELE UNUI TEXT S
I FRECVENTA LOR }
B1F8   { UTILIZIND UN POINTER CA FANION }
B1F8   PROGRAM FANION;
B1F8   TYPE TAB=ARRAY[1..10]OF CHAR;
B1F8        NOD=RECORD
B1F8              CHEIE:TAB;
B1F8              NUMAR:INTEGER;
B1F8              URM:^NOD
B1F8            END;
B1F8        POINT=^NOD;
B1F8   VAR CUV:TAB;
B201       INCEPUT,FANION:POINT;
B201   PROCEDURE CAUTA1(X:TAB;VAR INCEPUT:POINT);
B204   VAR Q:POINT;
B204   BEGIN
B21C    Q:=INCEPUT;
B22C    FANION^.CHEIE:=X;
B23D    WHILE Q^.CHEIE<>X DO
B261     Q:=Q^.URM;
```

```
B278   IF Q=FANION  {NU S-A GASIT}
B27F    THEN
B28D        BEGIN
B28D          Q:=INCEPUT;
B29D          NEW(INCEPUT);
B2A9          WITH INCEPUT^ DO
B2BB            BEGIN
B2BB             CHEIE:=X;
B2CC             NUMAR:=1;
B2DC             URM:=Q
B2E4            END
B2EF        END
B2EF    ELSE
B2F2        Q^.NUMAR:=Q^.NUMAR+1
B310  END; { CAUTA }
B31C  PROCEDURE AFISARE(Q:POINT);
B31F  VAR R:POINT;
B31F  BEGIN
B337   R:=Q;
B343   PAGE;
B348   WRITELN('CUVINTE          FRECVENTA');
B36E    WRITELN;
B371   WRITELN;
B374   WHILE R<>FANION DO
B38C    BEGIN
B38C        WRITELN(R^.CHEIE,'              ',R^.NUMAR);
B3C4        R:=R^.URM
B3CF    END
B3D8  END; { AFISARE }
B3E3  PROCEDURE PREGTIP;
B3E6  VAR I:INTEGER;
B3E6  BEGIN
B3FE   FOR I:=10 DOWNTO 1 DO
B420    IF CUV[I]=CHR(0)
B444        THEN
B44F            CUV[I]:=CHR(32)
B46F  END; { PREGTIP }
B47B  BEGIN   { MAIN }
B484   NEW(INCEPUT);
B48D   FANION:=INCEPUT;
B493   WRITELN('DATI CUVINTELE TEXTULUI SEPARATE PRIN <ENTER>(SFIRSIT=
ENTER)');
B4DD   READLN;
B4E0   READ(CUV);
B4E8   WHILE CUV[1]<>CHR(0) DO
B513    BEGIN
B513        PREGTIP;
B518        CAUTA1(CUV,INCEPUT);
B532        READLN;
B535        READ(CUV)
B53D    END;
B540   WRITELN(CHR(16));
B54A   AFISARE(INCEPUT);
B553   WRITELN(CHR(16))
B55A  END {$P}.
```

```
CUVINTE          FRECVENTA


PICIOARE             1
IN                   1
TOTUL                1
CALCA                1
VINE                 3
CAE                  1
NGROZITOAR           1
VIJELIA-             1
LUPTA                1
MINA-N               1
INSUSI               1
MIRCEA               1
BUCIUM               1
ARME                 1
ZGOMOT               1
DE                   3
CLOCOTI              1
CODRUL               1
ZBUCIUM              1
FREAMAT              1
MAI                  2
CE                   2
```

```
BATRINUL        1
PLECA           1
ABIA            1
SI              3
```

P.VI.6

6.2.6. Liste dublu înlănţuite

Unele aplicaţii necesită traversarea listelor şi inainte şi înapoi (adică atît de la primul element spre ultimul cît şi invers). Altfel pusă problema, fiind dat un element oarecare al listei, *trebuie determinat succesorul sau predecesorul acestuia.*

Dacă informaţia de legătură ataşată unui nod al listei se completează astfel încît pe lîngă cîmpul de tip pointer care indică următorul element din listă să mai existe un cîmp (tot de tip pointer) care să indice elementul din faţa nodului considerat, obţinem liste dublu înlănţuite.

Structura unei liste dublu înlănţuite este cea din fig.6.22.

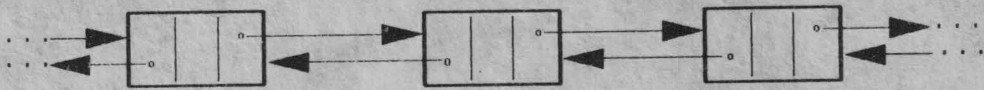


fig.6.22

Dacă implementarea acestor liste se realizează cu pointeri, în PASCAL HP4TM definirea listei este următoarea:

```
TYPE NOD=RECORD
           ELEMENT:...;
           ANTERIOR,URMATOR:^NOD
         END;
     POZITIE=^NOD;
```

Operaţiile prezentate şi exemplificate pe liste simplu înlănţuite (inserare, ştergere, căutare, traversare) se pot efectua şi pe liste dublu înlănţuite cu precizarea că ambele cîmpuri de legătură trebuie gestionate în mod adecvat.

Pentru exemplificare, vom prezenta modul de ştergere a elementului P^ dintr-o listă dublu înlănţuită. In fig.6.23 avem structura listei în care pointerul P indică un element oarecare (nici primul, nici ultimul).

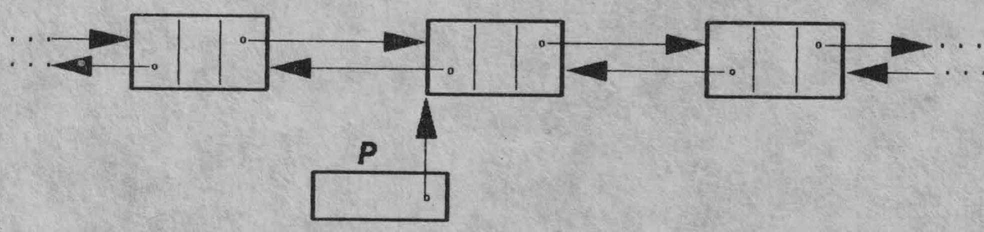


fig.6.23

Procedura PASCAL care realizează ştergerea nodului P^, este:

```
PROCEDURE SUPRIMA(VAR P:POZITIE);
BEGIN
  IF P^.ANTERIOR<>NIL
    THEN P^.ANTERIOR^.URMATOR:=P^.URMATOR;
  IF P^.URMATOR<>NIL
     THEN P^.URMATOR^.ANTERIOR:=P^.ANTERIOR;
END;       {SUPRIMA}
```

Efectul execuţiei procedurii SUPRIMA este vizualizat în fig.6.24.

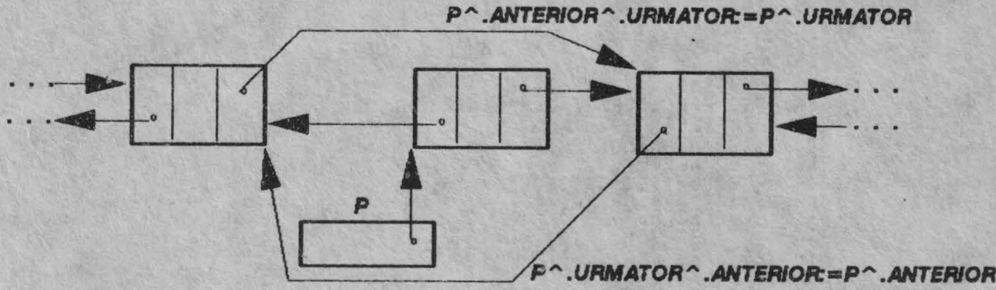


fig.6.24

Efectul primei instrucţiuni este localizarea pointerului care indică elementul care urmează să fie şters şi modificarea valorii sale astfel încît să indice succesorul nodului P^. In continuare se modifică cîmpul ANTERIOR al nodului care urmează celui indicat de P astfel încît el să indice nodul precedent celui indicat de P.

Nodul suprimat este indicat în continuare de P, spaţiul afectat lui putînd fi reutilizat în regim de alocare dinamică a memoriei, dacă în procedură se adaugă instrucţiunea

```
RELEASE(P);
```

Observaţie:

Lista dublu înlănţuită devine *circulară* prin introducerea informaţiilor de legătură care indică drept nod anterior al primului - ultimul nod şi drept nod succesor al ultimului - primul nod al listei. In acest caz nu mai este necesară (în procedura SUPRIMA) comparaţia cu NIL.

Programul **LISTADUBLINL** (P.VI.7) exemplifică operaţiile elementare (creare prin inserare, ştergere, căutare, traversare) posibile de efectuat şi pe liste dublu înlănţuite.

```
BA43  { CREAZA,AFISEAZA,CAUTA,SUPRIMA,INSEREAZA INTR-O LISTA DUBLU INL
ANTUITA }
BA43  PROGRAM LISTADUBLUINL;
BA43  TYPE TAB=ARRAY[1..20]OF CHAR;
BA43       NOD=RECORD
BA43            CHEIE:INTEGER;
BA43            NUME:TAB;
BA43            ANT,URM:^NOD
BA43           END;
BA43  VAR P,Q,R,T:^NOD;
BA4C      N,M,I,J,K,X:INTEGER;
BA4C      NUM:TAB;
BA4C      B:BOOLEAN;
BA4C      B:BOOLEAN;
BA4C  PROCEDURE CITNUM;
BA4F  BEGIN
BA67    WRITE('DATI NUMELE:');
BA7E    READLN;READ(NUM);
BA89  END; { CITNUME }
```

```
BA8F  PROCEDURE PREGTIP;
BA92  BEGIN
BAAA    FOR I:=20 DOWNTO 1 DO
BAC3       IF NUM[I]=CHR(0)
BAE4        THEN NUM[I]:=CHR(32)
BB0C  END; { PREGTIP }
BB17  PROCEDURE TRAVLISTA;
BB1A  BEGIN
BB32    T:=P;
BB38    I:=1;
BB3E    WHILE (T<>NIL) AND (I<=N) DO
BB67        BEGIN
BB67            WRITELN;
BB6A            WITH T^ DO
BB75                WRITELN(CHEIE,'   ',NUME);
BB9C            I:=I+1;
BBA3            T:=T^.URM
BBAB        END
BBB1  END; { TRAVLISTA }
BBBA  PROCEDURE INSEREAZAPRIMUL;
BBBD  BEGIN
BBD5   P:=NIL;
BBDB   NEW(Q);
BBE4   WRITE('DATI CHEIA:');
BBFA   READ(Q^.CHEIE);
BC06   P:=Q;
BC0C   P^.ANT:=NIL;
BC1C   Q^.URM:=NIL;
BC2C   CITNUM;
BC35   PREGTIP;
BC3E   Q^.NUME:=NUM
BC48  END; { INSEREAZAPRIMUL }
BC53  PROCEDURE INSEREAZAREST;
BC56  BEGIN
BC6E   NEW(R);
BC77   R^.URM:=NIL;
BC87   R^.ANT:=Q;
BC97   Q^.URM:=R;
BCA7   WRITE('DATI CHEIA:');
BCBD   READ(R^.CHEIE);
BCC9   CITNUM;
BCD2   PREGTIP;
BCDB   R^.NUME:=NUM;
BCEA   Q:=R
BCEA  END; { INSEREAZAREST }
BCF6  PROCEDURE MENIU;
BCF9  BEGIN
BD11   WRITELN('1. CREARE LISTA');
BD2E   WRITELN('2. AFISARE LISTA');
BD4C   WRITELN('3. INSERARE DUPA UN NOD');
BD71   WRITELN('4. INSERARE INAINTEA UNUI NOD');
BD9C   WRITELN('5. CAUTARE NOD DUPA CHEIE');
BDC3   WRITELN('6. SUPRIMARE NOD');
BDE1   WRITELN('7. STOP');
BDF6  END; { MENIU }
BDFC  PROCEDURE CREARE;
BDFF  BEGIN
BE17   WRITE('DATI NR.DE NODURI:');
BE34   READ(N);
BE3A   INSEREAZAPRIMUL;
BE43   IF N>1 THEN
BE56             FOR J:=2 TO N DO
BE74            INSEREAZAREST;
BE84  END; { CREARE }
BE8A  PROCEDURE STAI;
BE8D  BEGIN
BEA5   REPEAT UNTIL INCH<>CHR(0)
BEB0  END; { STAI }
BEC0  PROCEDURE CAUTA(VAR B:BOOLEAN);
BEC3  BEGIN
BEDB   B:=FALSE;
BEE5   T:=P;
BEEB   WHILE (T<>NIL) AND NOT B DO
BF0C    IF T^.CHEIE=X
BF14        THEN B:=TRUE
BF2C        ELSE T:=T^.URM
BF38  END; { CAUTA }
BF48  PROCEDURE AFISEAZA;
BF4B  BEGIN
BF63   PAGE;
BF68   WRITELN;
```

```
BF6B  IF N>0 THEN
BF7E        BEGIN
BF7E         WRITELN('LISTA ESTE URMATOAREA:');
BFA2         TRAVLISTA
BFA2        END
BFAB       ELSE
BFAE        WRITELN('LISTA VIDA!!!!')
BFC7 END; { AFISEAZA }
BFD0 PROCEDURE INSDNOD;
BFD3 BEGIN
BFEB  WRITE('DATI CHEIA DUPA CARE SE INSEREAZA:');
C018  READ(X);
C01E  CAUTA(B);
C02B  IF NOT B
C02B   THEN
C034       WRITE('CHEIA NU EXISTA!!!')
C051   ELSE
C054       BEGIN
C054          WRITE('DATI CHEIA:');
C06A          READ(M);
C070          CITNUM;
C079          PREGTIP;
C082          N:=N+1;
C089          NEW(R);
C092          R^.CHEIE:=M;
C09E          R^.NUME:=NUM;
C0AD          R^.URM:=T^.URM;
C0C5          R^.ANT:=T;
C0D5          T^.URM:=R
C0DD       END
C0E5 END; { INSDNOD }
C0EB PROCEDURE INSINOD;
C0EE BEGIN
C106  WRITE('DATI CHEIA INAINTEA CAREIA SE INSEREAZA:');
C139  READ(X);
C13F  CAUTA(B);
C14C  IF NOT B
C14C   THEN
C155       WRITE('CHEIA NU EXISTA!!!')
C172   ELSE
C175       BEGIN
C175          WRITE('DATI CHEIA:');
C18B          READ(M);
C191          CITNUM;
C19A          PREGTIP;
C1A3          N:=N+1;
C1AA          NEW(R);
C1B3          R^.CHEIE:=M;
C1BF          R^.NUME:=NUM;
C1CE          IF T^.ANT=NIL
C1DE            THEN
C1E8              BEGIN
C1E8               R^.URM:=T;
C1F8               T^.ANT:=R;
C208               R^.ANT:=NIL;
C218               P:=R
C218              END
C21E            ELSE
C221              BEGIN
C221               R^.URM:=T;
C231               T^.ANT^.URM:=R;
C249              END
C249       END
C249 END; { INSINOD }
C24F PROCEDURE CAUTARE;
C252 BEGIN
C26A  WRITE('DATI CHEIA DORITA:');
C287  READ(X);
C28D  CAUTA(B);
C29A  IF NOT B
C29A   THEN
C2A3       WRITELN('CHEIA NU EXISTA!!!')
C2C0   ELSE
C2C6       WRITELN('NODUL ESTE: ',T^.CHEIE,' ',T^.NUME);
C2FC END; { CAUTARE }
C302 PROCEDURE SUPRIMARE;
C305 BEGIN
C31D  WRITE('DATI CHEIA NODULUI CE SE VA SUPRIMA:');
C34C  READ(X);
C352  CAUTA(B);
```

```
C35F   IF NOT B
C35F    THEN
C368        WRITE('CHEIA NU EXISTA!!!')
C385    ELSE
C388        IF T^.ANT<>NIL THEN T^.ANT^.URM:=T^.URM ELSE P:=T^.URM;
C3D3        IF T^.URM<>NIL THEN T^.URM^.ANT:=T^.ANT ELSE T^.ANT^.URM:=
NIL;
C428   IF B THEN N:=N-1
C437  END; { SUPRIMA }
C43C
C43C  BEGIN  { MAIN }
C445   REPEAT
C445    STAI;
C44D    PAGE;
C452    WRITELN(CHR(18),CHR(1),CHR(22),CHR(5),CHR(14),'OPTIUNI',CHR(18
),CHR(0));
C498    MENIU;
C49D    WRITE(CHR(22),CHR(21),CHR(10),'ALEGETI OPTIUNEA:');
C4CE    READ(K);
C4D4    CASE K OF
C4D7        1:CREARE;
C4E9        2:AFISEAZA;
C4FB        3:BEGIN INSDNOD;STAI END;
C512        4:INSINOD;
C524        5:CAUTARE;
C536        6:SUPRIMARE;
C548        7:HALT
C552    END {CASE}
C555   UNTIL K=7
C55D  END {#P} .
```

P.VI.7

6.2.7. Structuri de date de tip stivă

O stivă este un tip special de listă, în care inserările şi suprimările se aplică la un singur capăt, care se numeşte vîrful stivei.

Stivele se mai numesc structuri listă de tip "LIFO" (LAST-IN - FIRST-OUT), adică "ultimul sosit - primul servit". Modelul intuitiv al unei stive este acela al unui vraf de farfurii pe o masă, în care maniera cea mai convenabilă de a lua un obiect sau de a adăuga un altul este, din motive lesne de înţeles, aceea de a acţiona în vîrful vrafului.

Asupra tipului de date de tip stivă putem defini următoarele proceduri care vor fi utilizate în programul STIVA (P.VI.8).

1) INITIALIZARE(S)-creează stiva vidă S;
2) VIRFST(S) -furnizează elementul din vîrful stivei S;
3) POP(S) -suprimă elementul din vîrful stivei; de regulă este convenabil ca funcţia POP să returneze elementul suprimat, necesitînd astfel o implementare proprie (POP(S,X));
4) PUSH(X,S) -introduce elementul X în vîrful stivei S;
5) STIVID(S) -returnează valoarea adevărat dacă S este vidă şi fals în caz contrar.

Utilizarea deosebit de frecventă şi cu mare eficienţă a structurii de date de tip stivă în domeniul programării, a generat implementarea hardware a acestui tip de structură în toate sistemele de calcul moderne şi includerea operatorilor specifici acestei structuri în setul de instrucţiuni cablate.

```
B472  { CREARE,AFISARE,TIPARIRE VIRF,SCOATERE ELEMENT DIN STIVA }
B472  PROGRAM STIVA;
B472  TYPE STIVA=RECORD
B472                 ELEM:INTEGER;
B472                 URM:^STIVA
B472              END;
B472       POINT=^STIVA;
B472  VAR VIRF,P:POINT;
B47B      STIVAB:STIVA;
B47B      OPT:CHAR;
B47B      OPTIUNI:SET OF CHAR;
B47B      I,N,W:INTEGER;
B47B  PROCEDURE INIT(VAR S:STIVA);
B47E  BEGIN
B496    NEW(VIRF);
B49F    VIRF^.URM:=NIL
B4A9  END; { INIT }
B4B4  FUNCTION STIVID(S:STIVA):BOOLEAN;
B4B7  BEGIN
B4CF    IF VIRF^.URM=NIL
B4DD        THEN
B4E7            STIVID:=TRUE
B4EA        ELSE
B4EF            STIVID:=FALSE
B4F1  END; { STIVID }
B4FD  FUNCTION VIRFST(S:STIVA):INTEGER;
B500  BEGIN
B518    IF STIVID(S)
B52E        THEN
B53D            WRITE('STIVA VIDA')
B552        ELSE
B555            VIRFST:=VIRF^.ELEM
B559  END; { VIRFST }
B56C  PROCEDURE POP(VAR S:STIVA);
B56F  BEGIN
B587    VIRF:=VIRF^.URM
B58D  END; { POP }
B59A  PROCEDURE PUSH(X:INTEGER;VAR S:STIVA);
B59D  BEGIN
B5B5    NEW(P);
B5BE    P^:=VIRF^;
B5CB    VIRF^.URM:=P;
B5D9    VIRF^.ELEM:=X
B5DD  END; { PUSH }
B5F2  PROCEDURE LISTARE(S:STIVA);
B5F5  BEGIN
B60D    P:=VIRF;
B613    IF P^.URM=NIL
B621        THEN
B62B            WRITE('STIVA VIDA!!!')
B643        ELSE
B646            WHILE P^.URM<>NIL DO
B661                  BEGIN
B661                    WRITE(P^.ELEM);
B66E                    P:=P^.URM
B674                  END
B67A  END; { LISTARE }
B687  PROCEDURE MENIU;
B68A  BEGIN
B6A2    WRITELN(CHR(22),CHR(10),CHR(5),'OPTIUNI');
B6CC    WRITELN;WRITELN;
B6D2    WRITELN('[P]UNERE IN STIVA');
B6F1    WRITELN('[S]COATERE DIN STIVA');
B713    WRITELN('[L]ISTARE STIVA');
B730    WRITELN('[T]IPARIRE VIRF STIVA');
B753    WRITELN('[E]ND');
B766    REPEAT UNTIL INCH IN OPTIUNI
B773  END; { MENIU }
B79B  BEGIN {MAIN}
B7A4    OPTIUNI:=['P','S','L','T','E'];
B7E0    INIT(STIVAB);
B7E9    REPEAT
B7E9        PAGE;
B7F1        REPEAT
B7F1          MENIU;
B7F9          FOR W:=1 TO 150 DO
B813        UNTIL INCH IN OPTIUNI;
B845    IF INCH='P'
B84C        THEN
B856            BEGIN
```

```
B856                  WRITE('CITE ELEMENTE?:');
B870                  READ(N);
B876                  FOR I:=1 TO N DO
B894                   BEGIN
B897                      WRITE('DATI ELEM.',I);
B8B5                      READ(STIVAB.ELEM);
B8C1                      PUSH(STIVAB.ELEM,STIVAB)
B8CD                   END
B8D2              END
B8D6          ELSE
B8D9              IF INCH='S'
B8E0                  THEN
B8EA                   BEGIN
B8EA                      WRITE('AM SCOS UN ELEMENT DIN STIVA!!!');
B914                      POP(STIVAB)
B918                   END
B91D          ELSE
B920              IF (INCH='T') AND NOT STIVID(STIVAB)
B940                  THEN
B94F                   BEGIN
B94F                    WRITELN;
B952                    WRITE('    VIRFUL STIVEI ESTE:');
B974                    WRITE(VIRFST(STIVAB));
B992                    REPEAT UNTIL INCH<>CHR(0)
B99D                   END
B9A7                  ELSE
B9AA                   IF INCH='L'
B9B1                      THEN
B9BB                       BEGIN
B9BB                        WRITELN;
B9BE                        WRITELN('       STIVA ESTE:');
B9DD                        LISTARE(STIVAB)
B9EE                       END;
B9F3       REPEAT UNTIL INCH IN ['G','E'];
BA24      UNTIL INCH='E'
BA2B  END {$P}.
```

P.VI.8

6.2.8. Structuri de date de tip coadă

Cozile sînt o altă categorie specială de liste în care elementele sînt inserate la un capăt (spate) şi sînt suprimate la celălalt capăt (faţă). Cozile se mai numesc liste "FIFO" (FIRST-IN - FIRST-OUT) adică de tipul "primul venit, primul servit".

Operaţiile care se pot efectua asupra cozii sînt analoage celor asupra stivei, cu deosebirea că inserările se fac la spatele cozii şi nu la începutul ei.

In programul **COADA** (P.VI.9) se vor utiliza următoarele proceduri:

1) INITIALIZARE(C) -creează coada vidă C;
2). FATZE(C) -este o funcţie care returnează primul element al cozii C;
3) ADAUGA(X,C) -inserează elementul X în spatele cozii;
4) SCOATE(C) -suprimă primul element al lui C;
5) VID(C) -returnează valoarea adevărat dacă C este vidă şi fals în caz contrar.

Observaţie:

Deoarece inserările se fac numai la spatele cozii, în procedura ADAUGA se va păstra un pointer la ultimul element. De asemenea, se va păstra şi pointerul la primul element al listei utilizat în execuţia procedurilor FATZE şi SCOATE. In implementare se poate utiliza un nod fictiv ca prim nod al cozii, caz în care pointerul de început va indica acest nod. Primul nod al cozii este unul fictiv în care cîmpurile informaţiilor propriu-zise sînt ignorate.

```
B3D9  PROGRAM COADA;
B3D9  TYPE NOD=RECORD
B3D9               ELEM:INTEGER;
B3D9               URM:^NOD
B3D9              END;
B3D9        PNOD=^NOD;
B3D9        COADA=RECORD
B3D9                   FATZA,SPATE:PNOD
B3D9                  END;
B3D9  VAR EL:INTEGER;
B3E2      COADAB:COADA;
B3E2      ER:BOOLEAN;
B3E2      I,N:INTEGER;
B3E2      A:CHAR;
B3E2  PROCEDURE INIT(VAR C:COADA);
B3E5  BEGIN
B3FD    NEW(C.FATZA);
B409    C.FATZA^.URM:=NIL;
B41E    C.SPATE:=C.FATZA;
B436  END; { INIT }
B43D  FUNCTION VID(C:COADA):BOOLEAN;
B440  BEGIN
B458    IF C.FATZA=C.SPATE
B46E        THEN
B47C             VID:=TRUE
B47F        ELSE
B484             VID:=FALSE
B486  END; { VID }
B492  FUNCTION FATZE(C:COADA):INTEGER;
B495  BEGIN
B4AD    IF VID(C)
B4C3        THEN
B4D2          BEGIN
B4D2             ER:=TRUE;
B4D7             WRITE('COADA VIDA!!!')
B4EF          END
B4EF        ELSE
B4F2          FATZE:=C.FATZA^.URM^.ELEM
B504  END; { FATZE }
B517  PROCEDURE ADAUGA(X:INTEGER;VAR C:COADA);
B51A  BEGIN
B532    NEW(C.SPATE^.URM);
B546    C.SPATE:=C.SPATE^.URM;
B566    C.SPATE^.ELEM:=X;
B57E    C.SPATE^.URM:=NIL
B591  END; { ADAUGA }
B59F  PROCEDURE SCOATE(VAR C:COADA);
B5A2  BEGIN
B5BA    IF VID(C)
B5CF        THEN
B5DE          BEGIN
B5DE             ER:=TRUE;
B5E3             WRITE('COADA VIDA!!!')
B5FB          END
B5FB        ELSE
B5FE          C.FATZA:=C.FATZA^.URM
B612  END; { SCOATE }
B621  PROCEDURE TIPARIRE(C:COADA);
B624  VAR P:PNOD;
B624  BEGIN
B63C    P:=C.FATZA;
B64D    P:=P^.URM;
B65F    WHILE P<>NIL DO
B677        BEGIN
B677         WRITE(P^.ELEM,' ');
B68C         P:=P^.URM
B695        END
B69E  END; { TIPARIRE }
B6AC  PROCEDURE MENIU;
B6AF  BEGIN
B6C7    WRITELN(CHR(22),CHR(10),CHR(10),'OPTIUNI');
B6F1    WRITELN;WRITELN;
B6F7    WRITELN('[A]DAUGARE IN COADA');
B718    WRITELN('[T]IPARIRE ELEMENT FATZA COADA');
B744    WRITELN('[S]COATERE ELEMENT DIN COADA');
B76E    WRITELN('[L]ISTARE COADA');
B78B    WRITELN('[O]PRIRE')
B79E  END; { MENIU }
```

```
B7A7  BEGIN { MAIN }
B7B0    INIT(COADAB);
B7B9    REPEAT
B7B9     PAGE;
B7C1     MENIU;
B7C6     READLN;
B7C9     READ(A);
B7CF     CASE A OF
B7D2
B7D2        'A':BEGIN
B7D7             WRITE('DATI NR.DE ELEMENTE:');
B7F6             READ(N);
B7FC             FOR I:=1 TO N DO
B81A              BEGIN
B81D               WRITELN('DATI ELEMENTUL:',I);
B843               READ(EL);
B849               ADAUGA(EL,COADAB)
B851              END
B856            END;
B85D        'S':SCOATE(COADAB);
B86E        'L':TIPARIRE(COADAB);
B88C        'O':HALT
B891      END;
B894    IF (A='T')AND NOT VID(COADAB)
B8B4       THEN
B8C3        WRITE('ELEMENTUL DIN FATZA ESTE:',FATZE(COADAB));
B905    REPEAT UNTIL INCH<>CHR(0);
B91A    UNTIL A IN ['O']
B92C END {$P}.
```

P.VI.9

6.2.9. Structura de date de tip arbore binar

Arborele, ca structură dinamică de date, s-a dovedit a fi util într-o gamă foarte variată de aplicaţii (căutări, sortări etc) unde se poate obţine economie de timp şi memorie, o minimizare a numărului de operaţii, ajungîndu-se la algoritmi eficienţi şi, implicit, la programe performante.

Prin *arbore binar* se înţelege o mulţime de n>=0 noduri care dacă nu este vidă, conţine un nod numit *rădăcină*, iar restul nodurilor formează doi arbori binari disjuncţi, numiţi: *subarborele stîng şi subarborele drept*.

In fig.6.25. sînt exemple de arbori binari distincţi:

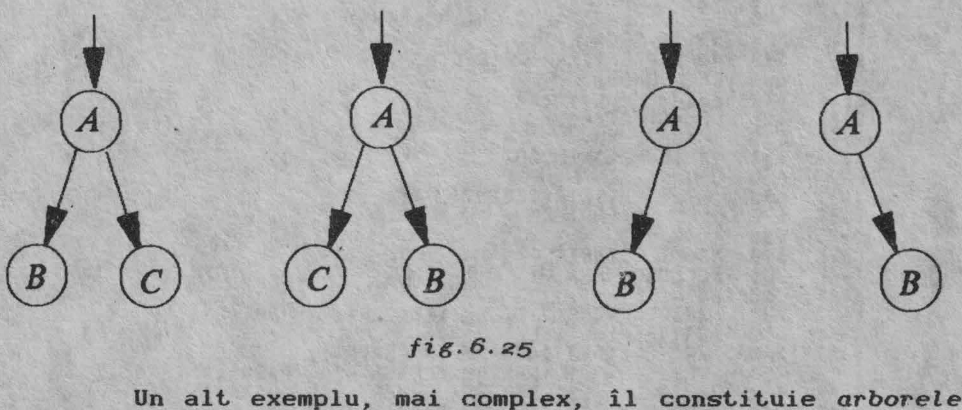


fig.6.25

Un alt exemplu, mai complex, îl constituie *arborele ataşat unei expresii aritmetice cu operatori binari*.

Fie spre exemplu expresia

(a+b/c)*(d-e*f)

căreia i se poate ataşa arborele binar din fig.6.26.

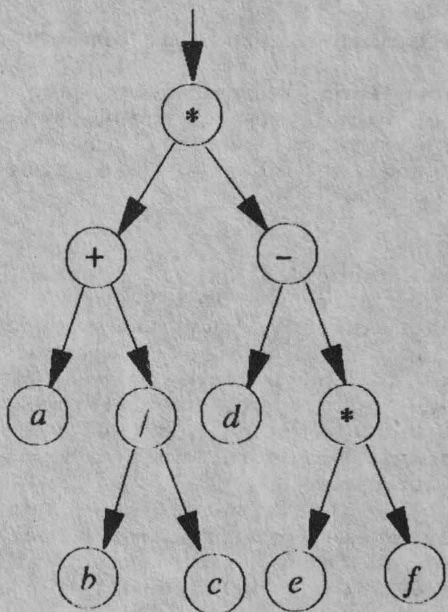


fig.6.26

Avînd la bază definiția recursivă a arborilor binari (paragraful 5.5), aceștia pot fi descriși în PASCAL HP4TM în felul următor:

```
TYPE NOD=RECORD
            INFO:...;
            STING,DREPT:^NOD
         END;
     ARBORE=^NOD;
```

Astfel, structura dinamică corespunzătoare arborelui din fig.6.26, în care cîmpul INFO este de tip CHAR, are configurația din fig.6.27.

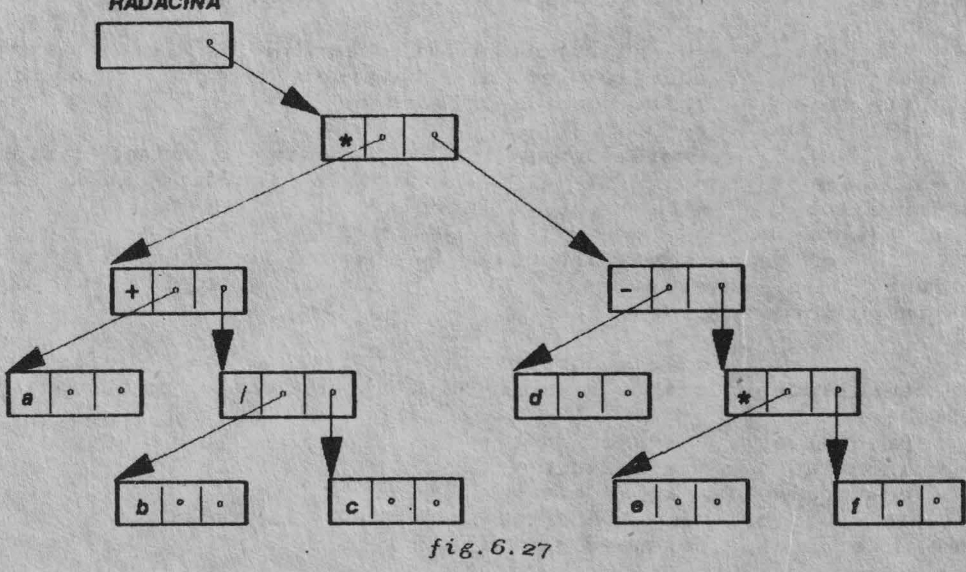


fig.6.27

Operaţii fundamentale asupra arborilor binari

Din multitudinea operaţiilor care se pot efectua asupra unui arbore binar vom prezenta cîteva, şi anume: *traversarea*, *căutarea (localizarea)*, *crearea*, *inserarea* şi *ştergerea* nodurilor unei structuri de tip arbore.

1. Traversarea

Traversarea arborilor binari constă în parcurgerea pe rînd a nodurilor arborelui în vederea efectuării unor prelucrări ale informaţiilor ataşate nodurilor. In timpul traversării, nodurile sînt *vizitate* într-o anumită ordine, astfel încît ele pot fi considerate ca şi cum ar fi integrate într-o listă liniară. De fapt, descrierea celor mai mulţi algoritmi este mult uşurată dacă în cursul prelucrării se poate preciza elementul următor al structurii arborescente, respectiv se poate liniariza arborele.

Algoritmul de traversare este foarte des utilizat deoarece majoritatea prelucrărilor care vizează *toate nodurile* unui arbore necesită un "drum" prin structură astfel încît să fie prelucrat *fiecare nod o singură dată*.

Există două posibilităţi de parcurgere a arborilor: *în adîncime* şi *în lăţime*, cel mai des folosită fiind cea în adîncime, care la rîndul ei poate fi de trei feluri. Ele sînt cunoscute sub denumirile de parcurgere în *inordine*, *preordine* şi *postordine*. Aceste denumiri sugerează poziţia rădăcinii în traversare (între cei doi descendenţi, înaintea lor, respectiv după ei).

Aceste moduri de traversare se definesc recursiv astfel:

-dacă arborele A este vid, atunci traversarea lui A generează o listă vidă;

-dacă A se reduce la un singur nod, atunci traversarea este banală, reducîndu-se la acest unic nod în oricare din cele trei moduri;

-pentru restul cazurilor, considerăm arborele A cu rădăcina R şi cu A1,A2 subarborii stîng, respectiv drept;

a) traversarea arborelui în *preordine* înseamnă lista formată din R, nodurile lui A1 traversat în preordine şi nodurile lui A2 traversat tot în preordine;

b) traversarea arborelui în *inordine* înseamnă lista formată din nodurile lui A1 vizitate în inordine , R şi nodurile lui A2 vizitate tot în inordine;

c) traversarea arborelui în *postordine* înseamnă lista obţinută prin traversarea lui A1 în postordine, apoi a lui A2 tot în postordine şi în final R.

Traversînd arborele din fig.6.27 şi afişînd caracterul corespunzător nodului vizitat, se obţin următoarele secvenţe:

```
preordine:    *+a/bc-d*ef;
inordine:     a+b/c*d-e*f;
postordine:   abc/+def*-*.
```

Ultima secvenţă este cunoscută în matematică sub numele de *notaţie poloneză postfixată*.

Observaţie:

Prin scrierea unei expresii aritmetice în notaţie poloneză se înţelege scrierea operatorului după ce doi operanzi, în loc de a-l scrie între ei ca în notaţia algebrică obişnuită (infixată).

Exemple:

```
a+b        devine ab+
a*b+c      devine ab*c+
a*(b+c)    devine abc+*
a-b/c      devine abc/-
(a-b)/c    devine ab-c/
```

Se observă că forma poloneză a unei expresii aritmetice se obţine parcurgînd în postordine arborele binar ataşat.

In *notaţia poloneză inversă (prefixată)* operatorul se scrie înaintea celor doi operanzi. De exemplu: a+b se scrie +ab. Se observă că notaţia prefixată corespunde traversării în preordine a arborelui expresiei.

O proprietate esenţială a notaţiei poloneze este aceea că la ambele forme ea păstrează semnificaţia expresiei aritmetice *fără a utiliza paranteze*.

In paragraful 5.5 au fost enumerate cîteva categorii de probleme pentru care se recomandă algoritmi recursivi. Printre ele s-a menţionat şi cazul problemelor care prelucrează structuri de date definite recursiv. Traversarea arborilor binari constituie un exemplu clasic pentru această categorie de probleme.

Cele trei metode de traversare le prezentăm în trei proceduri recursive în care R este o variabilă de tip pointer care indică rădăcina arborelui, iar PREL reprezintă operaţia care trebuie executată asupra fiecărui nod în parte.

Pentru structura de arbore în PASCAL HP4TM propunem următoarea definiţie:

```
TYPE NOD=RECORD
          INFO:...;
          STING,DREPT:^NOD
        END;
     REF=^NOD;
```

Procedurile corespunzătoare diferitelor moduri de traversare conţin o linie comentariu care trebuie înlocuită cu descrierea prelucrării concrete a nodului vizitat sau cu un apel al procedurii avînd acelaşi efect.

```
PROCEDURE PREORDINE(R:REF);
BEGIN
  IF R<>NIL THEN
             BEGIN
              {PREL(R^);}
               PREORDINE(R^.STING);
               PREORDINE(R^.DREPT)
             END;
END; {PREORDINE}
```

```
PROCEDURE INORDINE(R:REF);
BEGIN
  IF R<>NIL THEN BEGIN
                   INORDINE(R^.STING);
                   {PREL(R^);}
                   INORDINE(R^.DREPT)
                 END;
END; {INORDINE}

PROCEDURE POSTORDINE(R:REF);
BEGIN
  IF R<>NIL THEN BEGIN
                   POSTORDINE(R^.STING);
                   POSTORDINE(R^.DREPT);
                   {PREL(R^);}
                 END;
END; {POSTORDINE}
```

2. Căutarea

Problema *căutării* (localizării) unui nod se poate pune doar intr-un arbore binar de căutare. Prin *arbore binar de căutare* se înţelege un arbore binar care are proprietatea că, parcurgînd nodurile sale în inordine, secvenţa cheilor este monoton crescătoare (cheia este un cîmp unic de identificare a nodurilor în cadrul structurii arborescente). Intr-un un arbore binar de căutare cheile tuturor nodurilor din subarborele stîng al nodului *N* (avînd cheia *C*) au valori mai mici decît *C* şi toate nodurile din subarborele drept au valorile cheilor mai mari decît *C*. Din această proprietate rezultă un procedeu foarte simplu de căutare:

-se compară cheia rădăcinii (*C*) cu cheia căutată *X*; dacă C=X algoritmul se termină;
-dacă X<C se reia algoritmul pentru subarborele stîng;
-dacă X>C se reia algoritmul pentru subarborele drept.

Fie T un pointer care indică rădăcina unui arbore binar de căutare şi fie X un număr întreg dat. Atunci funcţia LOC(X,T) prezentată în continuare execută căutarea în arborele dat a nodului cu cheia egală cu X.

Această funcţie ia valoarea NIL dacă nu se găseşte nici un nod cu cheia X, altfel valoarea ei este egală cu pointerul care indică acest nod.

```
TYPE NOD=RECORD
          CHEIE:INTEGER;
          STING,DREPT:^NOD
        END;
     REF=^NOD;
FUNCTION LOC(X:INTEGER;VAR T:REF):REF;
VAR GASIT:BOOLEAN;
BEGIN
  GASIT:=FALSE;
  WHILE (T<>NIL) AND NOT GASIT DO
    IF T^.CHEIE=X
      THEN GASIT:=TRUE
      ELSE IF T^.CHEIE<X
             THEN T:=T^.STING
             ELSE T:=T^.DREPT;
  LOC:=T
END; {LOC}
```

Tehnica de căutare se poate simplifica dacă se aplică *metoda fanionului*. Aceasta constă în completarea arborelui cu un nod fictiv (fanion) indicat de un pointer notat cu F. In continuare structura arborelui se modifică înlocuind cu F toate referinţele egale cu NIL. Inainte de demararea procesului de căutare propriu-zisă, cheia fanionului se iniţializează cu X. In procesul căutării, nodul cu cheia X se găseşte acum cu certitudine; dacă acest nod este fanionul, atunci în arbore nu există un nod cu cheia X, în caz contrar, nodul găsit este cel căutat.

Pentru structura astfel modificată, funcţia LOC devine LOC1:

```
FUNCTION LOC1(X:INTEGER;VAR T:REF):REF;
BEGIN
  F^.CHEIE:=X;
  WHILE T^.CHEIE<>X DO
    IF X<T^.CHEIE
      THEN T:=T^.STING
      ELSE T:=T^.DREPT;
  LOC1:=T
END; {LOC1}
```

3. Crearea

Procesul de creare constă în inserarea a cîte unui nod într-un arbore binar de căutare iniţial vid. Problema care se pune este de a executa inserarea astfel încît arborele să rămînă ordonat şi după adăugarea noului nod. Aceasta se realizează traversînd arborele începînd cu rădăcina şi selectînd fiul stîng sau drept, după cum cheia de inserat este mai mică sau mai mare decît cea a nodului parcurs. Aceasta se repetă pînă cînd se ajunge la un pointer NIL. In continuare inserarea se realizează modificînd acest pointer astfel încît să indice noul nod. Precizăm că inserarea se poate realiza chiar dacă arborele conţine deja un nod cu cheia egală cu cea nouă, iar în cazul în care nu se doreşte introducerea acestuia în structură, în loc de inserare se poate emite un mesaj de forma "Nodul cu cheia ... există".

In continuare se va prezenta o procedură recursivă de inserare a unui nod într-un arbore binar de căutare.

Dacă X este un număr întreg reprezentînd cheia nodului de inserat şi T un pointer care indică rădăcina arborelui, atunci următoarea procedură (INARBORE) realizează inserarea nodului cu cheia X.

Se observă că pentru funcţionarea corectă a acestei proceduri este esenţial ca T să fie parametru transmis prin referinţă, deoarece noua valoare pe care o primeşte T prin instrucţiunea NEW(T) numai astfel se transmite parametrului actual corespunzător.

```
PROCEDURE INARBORE(X:INTEGER;VAR T:REF);
BEGIN
  IF T<>NIL
    THEN [*]
      IF X<T^.CHEIE
        THEN
          INARBORE(X,T^.STING)
        ELSE
          INARBORE(X,T^.DREPT)
```

```
        ELSE
          BEGIN
            NEW(T);
            WITH T^ DO
              BEGIN
                CHEIE:=X;
                STING:=NIL;  [**]
                DREPT:=NIL   [***]
              END
          END
  END; {INARBORE}
```

Considerînd un arbore de căutare cu structura din fig.6.28, efectul procedurii INARBORE pentru inserarea nodului cu cheia 8 este vizualizat în fig.6.29.

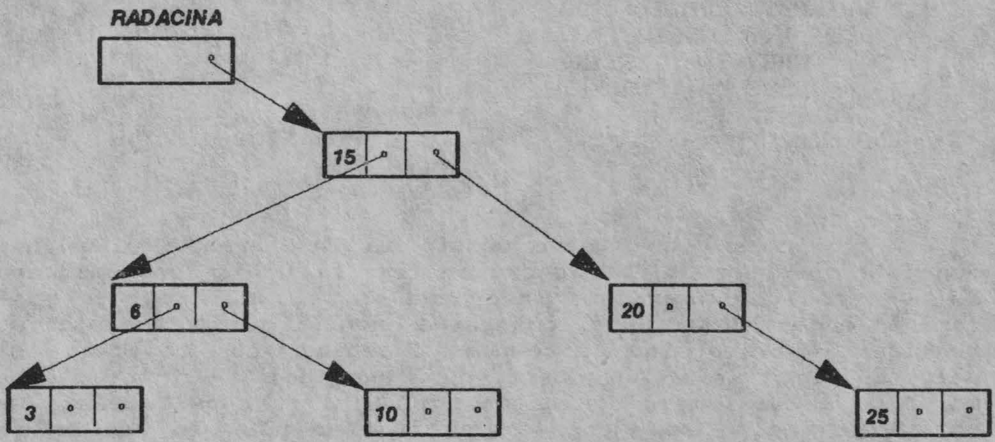


fig.6.28

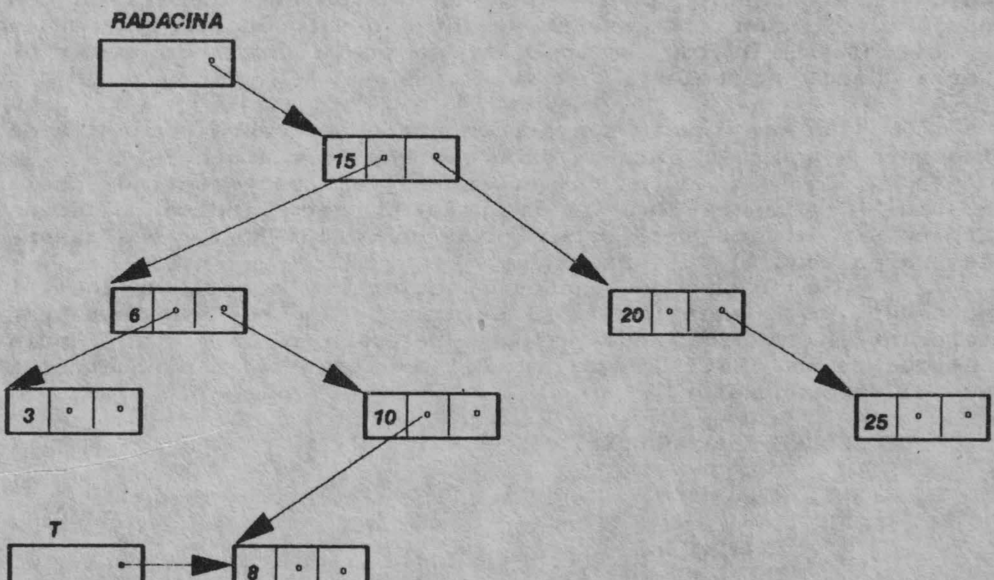


fig.6.29

In continuare se prezintă un fragment de program principal care apelează procedura de mai sus. Se presupune că toate cheile sînt diferite de zero, se introduc de la tastatură şi sînt urmate de o cheie *fictivă* egală cu zero, valoare care încheie secvenţa cheilor.

```
VAR RADACINA:REF;
    C:INTEGER;
BEGIN
  RADACINA:=NIL; [*]
  READ(C);
  WHILE C<>0 DO
    BEGIN
      INARBORE(C,RADACINA);
      WRITE('DATI CHEIA:');
      READ(C)
    END
END;
```

Utilizarea metodei fanionului în crearea arborilor binari de căutare presupune următoarele modificări:

-în secvenţa de program principal se declară variabila F:REF, iar instrucţiunea RADACINA:=NIL marcată cu [*] în secvenţa de instrucţiuni se înlocuieşte cu NEW(F); RADACINA:=F; servind la iniţializarea rădăcinii. In procedura INARBORE, condiţia insrucţiunii IF din linia marcată cu [*] (T<>NIL) se înlocuieşte cu T<>F, iar instrucţiunile STING:=NIL şi DREPT:=NIL marcate cu [**] respectiv [***] se înlocuiesc cu STING:=F; şi DREPT:=F;.

4. Ştergerea

Se consideră o structură de arbore binar de căutare şi X o cheie precizată. Se cere să se suprime din structura de arbore nodul avînd cheia X. Pentru aceasta, în prealabil, se caută dacă există un nod cu o astfel de cheie. Dacă nu, suprimarea se consideră încheiată şi eventual se emite un mesaj. In caz contrar se execută suprimarea propriu-zisă, de o asemenea manieră încît arborele să rămînă arbore de căutare şi după eliminarea nodului respectiv.

Se disting două cazuri, după cum nodul care trebuie suprimat are *cel mult un fiu sau doi fii*.

1.Primul caz se rezolvă conform fig.6.30-6.35 în care se prezintă cele trei alternative posibile:

a)-nodul avînd cheia X are doar subarbore sting;
b)-nodul avînd cheia X are doar subarbore drept;
c)-nodul avînd cheia X nu are subarbori.

Pentru ştergerea nodului cu cheia X,în aceste cazuri, notăm cu P pointerul corespunzător cîmpului de referinţă al predecesorului (tatălui) nodului avînd cheia X, care indică acest nod. valoarea lui P se modifică astfel încît acesta să indice unicul fiu al lui X (dacă acesta există) sau, în caz contrar, P devine NIL.

Fragmentul de program care realizează această operaţie este următorul:

```
Q:=P;                      {P indică nodul cu cheia X}
IF Q^.DREPT=NIL THEN P:=Q^.STING
                ELSE IF Q^.STING=NIL
                       THEN  P:=Q^.DREPT;
```

Pentru a putea urmări efectul secvenţei anterioare, în cazul a), considerăm fig.6.30 cu un fragment de arbore.

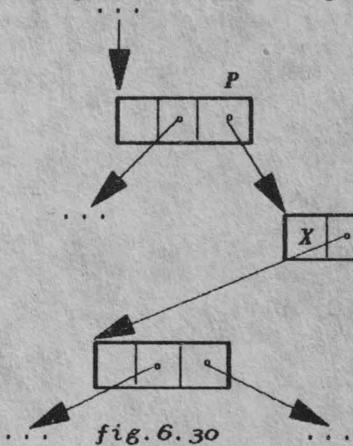


fig. 6. 30

După execuţia instrucţiunilor specificate, structura arborelui este cea din fig.6.31. Spaţiul alocat nodului avînd cheia X poate fi redat memoriei prin apelul procedurii *RELEASE*.

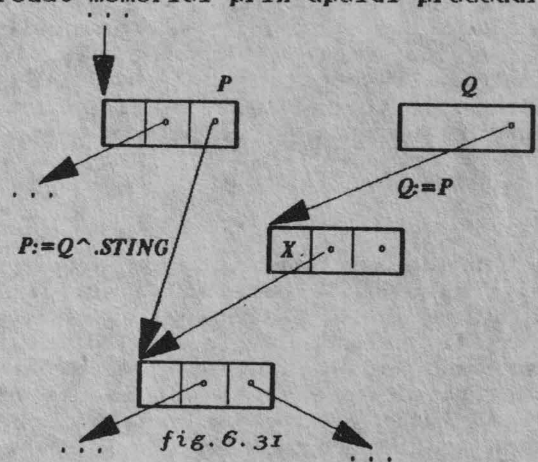


fig. 6. 31

In cazul b), pornim de la fragmentul de arbore din fig.6.32 care după efectuarea suprimării nodului avînd cheia X va avea structura din fig.6.33.

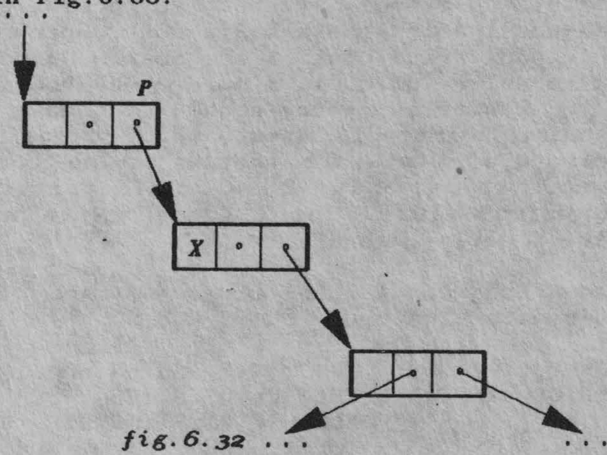


fig. 6. 32

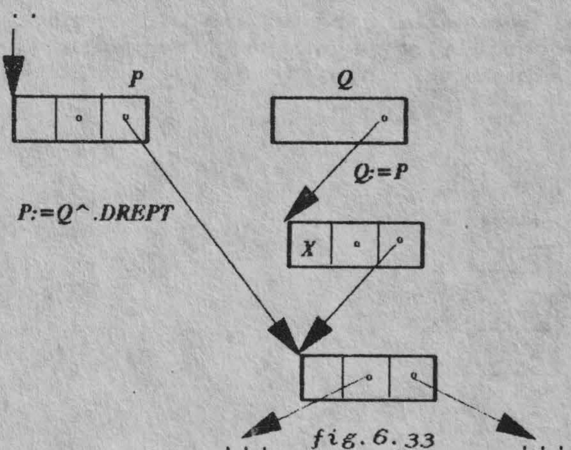


fig.6.33

Cazul c) este redat grafic în fig.6.34 şi 6.35.

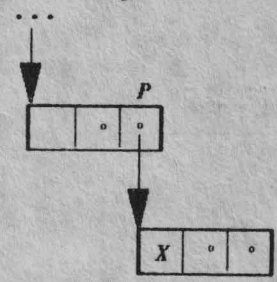


fig.6.34

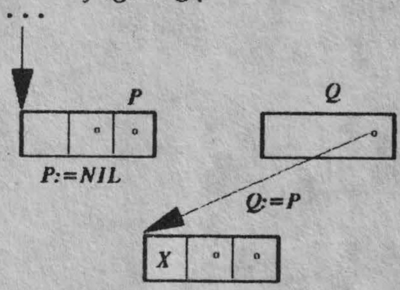


fig.6.35

2. Pentru a trata cazul în care nodul cu cheia X are doi fii, considerăm fragmentul de arbore din fig.6.36.

Suprimarea nodului X în acest caz necesită localizarea celui mai din dreapta fiu al subarborelui stîng sau a celui mai din stînga fiu al subarborelui drept. Primul are cheia precedentă cheii X, al doilea are cheia imediat următoare în secvenţa ordonată a cheilor, deci pentru ca arborele să rămînă arbore de căutare, unul din cele două noduri (în general cel mai din dreapta fiu al subarborelui stîng), fie acesta nodul cu cheia Y, trebuie adus în locul nodului suprimat. După ce informaţiile propriu-zise ale nodului care trebuie suprimat se înlocuiesc cu informaţiile propriu-zise ale nodului cu cheia Y, vechiul nod cu cheia Y (care nu are fiu drept) se suprimă ca în cazul 1 (a).

Nodul Y se detectează după următoarea metodă: se construieşte o secvenţă de noduri care începe cu fiul stîng al lui X, după care se alege drept succesor al fiecărui nod, fiul său

drept. Primul nod al secvenţei care nu are fiu drept este Y.

In urma operaţiilor descrise, fragmentul de arbore din fig.6.36 va avea structura din fig.6.37.

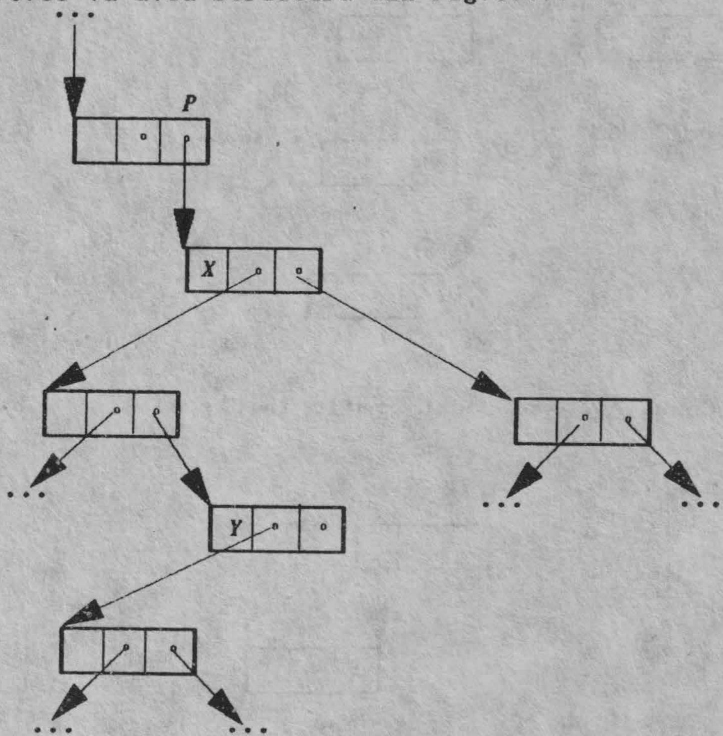


fig.6.36

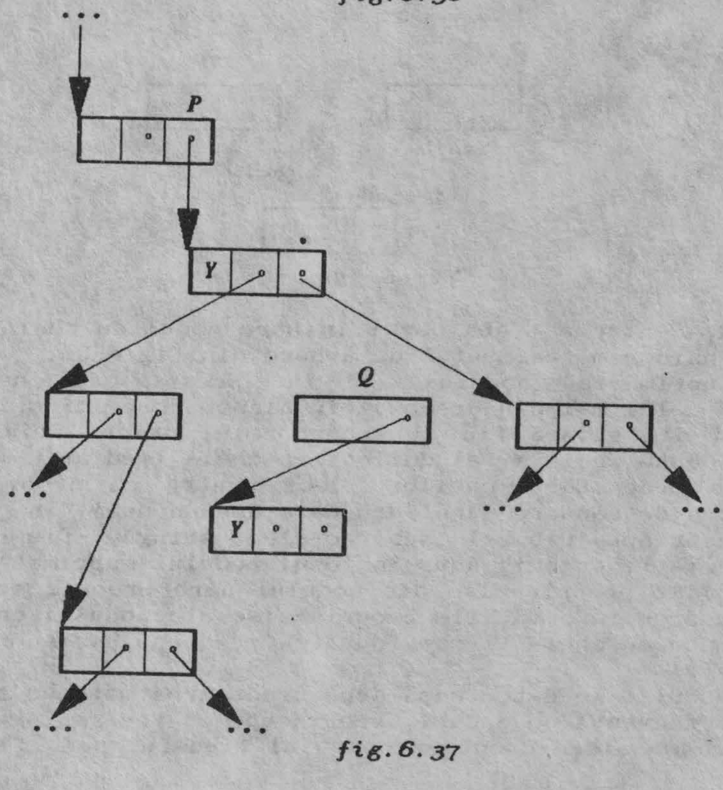


fig.6.37

Suprimarea unui nod dintr-un arbore binar de căutare se realizează cu procedura SUPRIMARE. Procedura locală SUPRED caută predecesorul în inordine al nodului cu cheia X, realizînd suprimarea acestuia conform metodei descrise. Procedura SUPRED se utilizează numai în situaţia în care nodul cu cheia X are doi fii.

```
PROCEDURE SUPRIMARE(X:INTEGER;VAR F:REF);
VAR Q:REF;

PROCEDURE SUPRED(VAR R:REF);
BEGIN
  IF R^.DREPT<>NIL
    THEN SUPRED(R^.DREPT)
    ELSE
      BEGIN
        Q^.CHEIE:=P^.CHEIE;
        Q^.NUMAR:=P^.NUMAR;
        Q:=R;
        R:=R^.STING
      END
END; {SUPRED}

BEGIN  {SUPRIMARE}
  IF P=NIL
    THEN WRITELN('NOD INEXISTENT')
    ELSE IF X<P^.CHEIE
           THEN
             SUPRIMARE(X,P^.STING)
           ELSE
             IF X>P^.CHEIE
               THEN
                 SUPRIMARE(X,P^.DREPT)
               ELSE
                 BEGIN
                   Q:=P;
                   IF Q^.DREPT=NIL
                     THEN
                       P:=Q^.STING
                     ELSE
                       IF Q^.STING=NIL
                         THEN
                           P:=Q^.DREPT
                         ELSE
                           SUPRED(Q^.STING);
                   [*]
                 END
END; {SUPRIMARE}
```

Dacă în locul marcajului [*] apelăm procedura RELEASE pentru pointerul Q, spaţiul alocat nodului suprimat se redă memoriei.

In final, vom ilustra modul în care se suprimă anumite noduri dintr-un arbore de căutare prin apelul procedurii SUPRIMARE. In fig.6.38 am pus în evidenţă doar cîmpul cheie ataşat nodului, iar nodul care urmează să fie şters este marcat cu o săgeată.

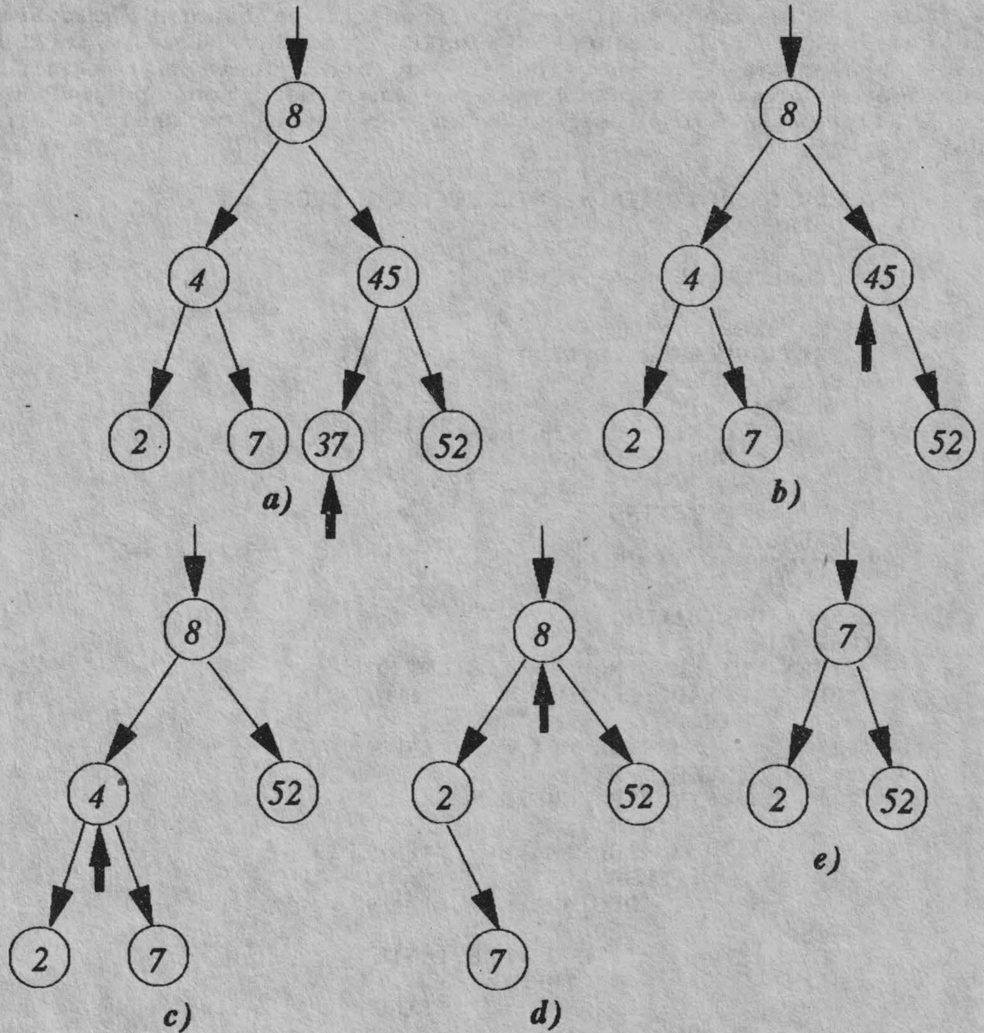


fig. 6.38

Programul **ARBOCOM** (P.VI.10) creează un arbore binar de căutare (arbore binar ordonat) şi execută asupra lui operaţiile prezentate în acest subparagraf.

```
BA3B  PROGRAM ARBOCOM;
BA3B  TYPE NOD=RECORD
BA3B          CHEIE:CHAR;
BA3B          STING,DREPT:^NOD
BA3B         END;
BA3B       POINT=^NOD;
BA3B  VAR R,P,FANION,Q:POINT;
BA44      N,I:INTEGER;.
BA44      A,X:CHAR;
BA44      OPT:SET OF CHAR;
BA44  PROCEDURE PREORDINE(R:POINT);
BA47  BEGIN
BA5F    IF R<>NIL
BA6A     THEN
BA74       BEGIN
BA74        WRITE(R^.CHEIE,' ');
BA83        PREORDINE(R^.STING);
BA98        PREORDINE(R^.DREPT)
BAA6       END
BAAF  END; { PREORDINE }
```

```
BAB6  PROCEDURE INORDINE(R:POINT);
BAB9  BEGIN
BAD1    IF R<>NIL
BADC     THEN
BAE6        BEGIN
BAE6         INORDINE(R^.STING);
BAFB         WRITE(R^.CHEIE);
BB05         INORDINE(R^.DREPT)
BB13        END
BB1C  END; { INORDINE }
BB23  PROCEDURE POSTORDINE(R:POINT);
BB26  BEGIN
BB3E    IF R<>NIL
BB49     THEN
BB53        BEGIN
BB53         POSTORDINE(R^.STING);
BB68         POSTORDINE(R^.DREPT);
BB7F         WRITE(R^.CHEIE)
BB89        END
BB89  END; { POSTORDINE }
BB90  PROCEDURE CREARE(VAR X:CHAR;VAR P:POINT);
BB93  BEGIN
BBAB    READLN;
BBAE    READ(X);
BBBA    IF X<>'.'
BBC5     THEN
BBCE        BEGIN
BBCE         NEW(P);
BBDA         P^.CHEIE:=X;
BBEE         CREARE(X,P^.STING);
BC0A         CREARE(X,P^.DREPT)
BC1F        END
BC28     ELSE
BC2B         P:=NIL
BC36  END; { CREARE }
BC44  FUNCTION CAUTA(X:CHAR;VAR G:POINT):POINT;
BC47  VAR GASIT:BOOLEAN;
BC47  BEGIN
BC5F    GASIT:=FALSE;
BC63    WHILE (G<>NIL) AND NOT GASIT DO
BC87      IF G^.CHEIE=X
BC93        THEN
BCA1         GASIT:=TRUE
BCA4        ELSE
BCA9         IF G^.CHEIE<X
BCB5          THEN
BCC2            G:=G^.STING
BCD5          ELSE
BCE0            G:=G^.DREPT;
BD00    CAUTA:=G
BD00  END; { CAUTA }
BD19  FUNCTION CAUTACUFANION(X:CHAR;VAR T:POINT):POINT;
BD1C  BEGIN
BD34    FANION^.CHEIE:=X;
BD3D    WHILE T^.CHEIE<>X DO
BD59     IF X<T^.CHEIE
BD68      THEN
BD72        T:=T^.STING
BD85      ELSE
BD90        T:=T^.DREPT;
BDB0    CAUTACUFANION:=T
BDB0  END; { CAUTACUFANION }
BDC8  PROCEDURE CREARBINORD(X:CHAR;VAR T:POINT);
BDCB  BEGIN
BDE3    IF T<>NIL
BDF2     THEN
BDFC      IF X<T^.CHEIE
BE0B       THEN
BE15        CREARBINORD(X,T^.STING)
BE26       ELSE
BE32        CREARBINORD(X,T^.DREPT)
BE45     ELSE
BE51      BEGIN
BE51       NEW(T);
BE5D       WITH T^ DO
BE6F        BEGIN
BE6F         CHEIE:=X;
BE78         STING:=NIL;
BE85         DREPT:=NIL
BE90        END
BE94      END
BE94  END; { CREARBINORD }
```

```
BE9C PROCEDURE SUPRIMARE(X:CHAR;VAR P:POINT);
BE9F VAR Q:POINT;
BE9F   PROCEDURE SUPRED(VAR R:POINT);
BEA2   BEGIN
BEBA       IF R^.DREPT<>NIL
BED0        THEN
BEDA         SUPRED(R^.DREPT)
BEE8        ELSE
BEF4         BEGIN
BEF4          Q^.CHEIE:=P^.CHEIE;
BF16          Q:=R;
BF2F          R:=R^.STING
BF42         END
BF4A   END; { SUPRED }
BF51 BEGIN { SUPRIMARE }
BF69-  IF P=NIL
BF78    THEN
BF82     WRITELN('NODUL NU SE GASESTE!')
BFA1    ELSE
BFA7     IF X<P^.CHEIE
BFB6      THEN
BFC0       SUPRIMARE(X,P^.STING)
BFD1      ELSE
BFDD       IF X>P^.CHEIE
BFEC        THEN
BFF5         SUPRIMARE(X,P^.DREPT)
C008        ELSE
C014         BEGIN
C014          Q:=P;
C024          IF Q^.DREPT=NIL
C036           THEN
C040            P:=Q^.STING
C04F           ELSE
C05A            IF Q^.STING=NIL
C06A             THEN
C074              P:=Q^.DREPT
C085             ELSE
C090              SUPRED(Q^.STING);
C09D          RELEASE(Q)
C0A5         END
C0AB END; { SUPRIMARE }
C0B4 PROCEDURE MENIU;
C0B7 BEGIN
C0CF   WRITELN(CHR(22),CHR(10),CHR(5),'OPTIUNI');
C0F9   WRITELN('[C]REARE ARBORE BINAR');
C11C   WRITELN('[A]RBORE BINAR ORDONAT');
C140   WRITELN('[T]IPARIRE NOD ARBORE BINAR');
C169   WRITELN('[L]ISTARE ARBORE BINAR');
C18D   WRITELN('[S]UPRIMARE NOD');
C1AA   WRITELN('[O]PRIRE');
C1C0   OPT:=['O','L','C','T','A','S'];
C203   REPEAT UNTIL INCH IN OPT
C210 END; { MENIU }
C238 BEGIN {MAIN}
C241   NEW(P);
C24A   REPEAT
C24A    PAGE;
C252    MENIU;
C257    READLN;
C25A    READ(A);
C260    CASE A OF
C263     'C':BEGIN
C268          WRITELN('DATI ELEMENTELE SEPARAT:');
C28E          CREARE(X,P);
C29B          WRITELN;
C29E          WRITELN('ARBORELE BINAR ESTE CREAT!')
C2C3         END;
C2C9     'A':BEGIN
C2CE          WRITELN('DATI ELEMENTELE SEPARAT:');
C2F4          P:=NIL;
C2FA          READLN;
C2FD          READ(X);
C303          WHILE X<>' ' DO
C316           BEGIN
C316            CREARBINORD(X,P);
C324            READLN;
C327            READ(X)
C32D           END;
C330          WRITELN;
C333          WRITELN('ARBORELE BINAR ESTE CREAT!')
C358         END;
```

```
C35E      'L':BEGIN
              WRITELN('1) PREORDINE');
C363
C37D          WRITELN('2) INORDINE');
C396          WRITELN('3) POSTORDINE');
C3B1          READLN;
C3B4          READ(A);
C3BA          CASE A OF
C3BD           '1':PREORDINE(P);
C3CE           '2':INORDINE(P);
C3DF           '3':POSTORDINE(P)
C3E8          END
C3ED         END;
C3F0      'T':BEGIN
C3F5          WRITELN('1) FARA FANION');
C411          WRITELN('2) CU FANION');
C42B          READLN;
C42E          READ(A);
C434          WRITELN('DATI CHEIA NODULUI:');
C455          READLN;
C458          READ(X);
C45E          CASE A OF
C461           '1':BEGIN
C466                R:=CAUTA(X,P);
C479                WRITELN(R^.CHEIE)
C480               END;
C486           '2':BEGIN
C48B                R:=CAUTACUFANION(X,P);
C49E                WRITELN(R^.CHEIE)
C4A5               END
C4A8          END
C4A8         END;
C4AB      'S':BEGIN
C4B0          WRITELN('DATI CHEIA NODULUI:');
C4D1          READLN;
C4D4          READ(X);
C4DA          SUPRIMARE(X,P)
C4E3         END
C4E8     END;
C4E8    READLN;
C4EB   UNTIL A='0';
C4FC  END {$P}.
```

P.VI.10

Observaţie:

Intr-o structură de arbore, nodurile se găsesc pe *niveluri* definite astfel: nivelul 1 îl formează rădăcina, succesorii rădăcinii (fiii ei) formează nivelul 2, în general fiii tuturor nodurilor nivelului *n* formează nivelul n+1. Nivelul maxim al nodurilor unui arbore se numeşte *înălţimea arborelui*.

In general, înălţimea unui arbore nu este determinată de numărul nodurilor sale. Este simplu de observat că un arbore are înălţimea minimă dacă fiecare nivel conţine numărul maxim de noduri posibile, cu excepţia ultimului nivel.

Deoarece numărul maxim de noduri al nivelului *i* este 2^{i-1}, rezultă că înălţimea minimă a unui arbore binar cu *n* noduri este $[\log_2 n]+1$.

Arborii binari de căutare, în general, nu respectă această cerinţă, deci o căutare necesită mai multe comparaţii, arborele avînd mai mult de $[\log_2 n]+1$ niveluri.

Dacă ordinea în care sînt plasate nodurile în arbore nu contează, dar se doreşte crearea unui arbore cu număr minim de niveluri, pentru construirea acestuia se poate alege o soluţie ca în programul **CONSTARBORE** (P.VI.11) în care se creează un arbore *perfect echilibrat*, care are proprietatea că diferenţa dintre numărul nodurilor din subarborii stîng, respectiv drept, este cel mult 1.

```
B08A  { CONSTRUIRE ARBORE PERFECT ECHILIBRAT }
B08A  PROGRAM CONSTARBORE;
B08A  TYPE NOD=RECORD
B08A            CHEIE:INTEGER;
B08A            STING,DREPT:^NOD
B08A          END;
B08A       POINT=^NOD;
B08A  VAR RADACINA:POINT;
B093      N:INTEGER;
B093  FUNCTION ARBORE(N:INTEGER):POINT;
B096  VAR NODNOU:POINT;
B096      X,NS,ND:INTEGER;
B096  BEGIN {CONSTRUIRE ARBORE ECHILIBRAT CU N NODURI}
B0AE    IF N=0
B0B9     THEN
B0C3        ARBORE:=NIL
B0C7     ELSE
B0CF        BEGIN
B0CF         NS:=N DIV 2;
B0E3         ND:=N-NS-1;
B0FF         WRITE('DATI X:');
B111         READ(X);
B11A         NEW(NODNOU);
B127         WITH NODNOU^ DO
B135          BEGIN
B135           CHEIE:=X;
B144           STING:=ARBORE(NS);
B161           DREPT:=ARBORE(ND)
B171          END;
B180         ARBORE:=NODNOU
B180        END
B18C  END; { ARBORE }
B198  PROCEDURE TIPARESTE(T:POINT;H:INTEGER);
B19B  VAR I:INTEGER;
B19B  BEGIN
B1B3    IF T<>NIL
B1BE        THEN
B1C8         WITH T^ DO
B1D6          BEGIN
B1D6           TIPARESTE(STING,H-5);
B1F9           FOR I:=1 TO H DO
B223                 WRITE(' ');
B22F           WRITELN(CHEIE);
B23F           TIPARESTE(DREPT,H-5)
B25B          END
B264  END; { TIPARESTE }
B26F  BEGIN {MAIN}
B278    WRITE('DATI NR.DE NODURI:');
B295    READ(N);
B29B    RADACINA:=ARBORE(N);
B2A9    TIPARESTE(RADACINA,20)
B2B1  END {$P}.
```

P.VI.11

Probleme propuse

1. Se consideră polinoame de forma:

$$P(x)=c_1x^{e_1}+c_2x^{e_2}+\ldots+c_nx^{e_n},\ \text{unde } e_i>=0,\ i=1,\ldots,n.$$

Un astfel de polinom poate fi reprezentat prin intermediul unei liste înlănţuite în care fiecare nod conţine trei cîmpuri: unul pentru coeficientul C_i, al doilea pentru exponentul e_i şi al treilea de tip pointer la nodul următor.

Se cere să se scrie un program care determină:

a) suma a două polinoame;
b) diferenţa a două polinoame;
c) produsul a două polinoame.

2. O coadă cu două capete (DEQUEUE) este o listă din care elementele pot fi suprimate sau inserate la ambele capete. Se cere să se creeze astfel de cozi utilizînd structuri de tip pointer.

3. Fiind dată o greutate totală GT şi un set de greutăţi reprezentate prin întregi pozitivi, a1,a2,...an, se cere să se afle dacă greutăţile se pot selecta astfel încît suma lor să fie exact GT (problema "sacului călătorului"). Se va utiliza structura de tip stivă.

4. Dîndu-se un text, să se afişeze cuvintele textului şi frecvenţa lor cu ajutorul structurilor de tip arbore.

5. Un arbore binar poate fi definit ca o structură abstractă de date formată din structura de arbore propriu-zisă căreia i se asociază un set de operatori specifici: FIUSTING(N), FIUDREPT(N), PARINTE(N) şi VID(N), unde N este cheia ataşată nodului. Primii trei operatori returnează fiul stîng, fiul drept respectiv părintele nodului (sau 0 dacă vreunul nu există) iar VID(N) returnează valoarea adevărat dacă şi numai dacă nu există nod cu cheia N.

Se cere să se implemnteze aceste proceduri utilizînd reprezentarea arborilor cu ajutorul tipului pointer.

6. Catalogul unei biblioteci este organizat ca o structură de arbore binar de căutare. Fiecare nod se referă la o carte şi conţine titlul cărţii, autorul, data intrării în bibliotecă şi data ultimului împrumut. Se cere să se redacteze un program care traversează arborele şi şterge toate cărţile împrumutate înaintea unei date precizate. Se precizează că datele se înregistrează ca numere întregi.

VII. ELEMENTE DE GRAFICA SI SUNET SPECIFICE LIMBAJULUI HP4TM

7.1. GENERALITATI

Este normal ca implementarea limbajului PASCAL pe calculatoare personale, care au facilităţi de grafică şi sunet, aşa cum sînt calculatoarele din familia Spectrum, să conţină şi aceste extensii referitoare la grafică şi sunet. Pentru a nu mări dimensiunea compilatorului şi editorului, s-a adoptat ideea folosirii funcţiilor implementate şi a rutinelor deja existente în memoria ROM a calculatorului. Intregul pachet de subprograme de grafică şi sunet este scris în limbajul PASCAL, el putînd fi încărcat de pe casetă şi adăugat oricărui program deja creat. Evident, setul de proceduri conţinut în pachetul numit TURTLE, este un set minimal, dar suficient; el poate fi modificat, îmbunătăţit, sau mărit prin adăugarea altor proceduri, de către utilizatori.

Ne propunem să discutăm fiecare procedură a pachetului iniţial, oferit de firma Hisoft şi să indicăm cîteva dezvoltări posibile.

Pachetul de subprograme iniţial este conceput în stil LOGO, deci cursorul (BROASCA - TURTLE) poate fi deplasat cu ajutorul unor comenzi foarte simple. Cursorul poate lăsa urme, sau poate fi făcut invizibil, adică poate să se deplaseze fără a lăsa urme. Poziţia şi orientarea cursorului se păstrează în variabile globale, care sînt actualizate cînd cursorul este deplasat sau rotit; evident aceste variabile pot fi consultate sau modificate în orice moment.

7.2. VARIABILE GLOBALE

Variabilele globale utilizate de către pachetul TURTLE sînt: HEADING, XCOR, YCOR, PENSTATUS.

HEADING

Această variabilă este utilizată pentru a păstra valoarea unghiulară a orientării cursorului. Variabila poate lua orice valoare reală (în grade) şi poate fi iniţializată cu zero prin procedura TURTLE. Valoarea 0 corespunde direcţiei EST, deci după apelarea procedurii TURTLE, cursorul va fi orientat spre dreapta. Cînd variabila HEADING creşte, cursorul se va roti în sens trigonometric, invers acelor de ceasornic.

XCOR, YCOR

Aceste variabile de tip real conţin coordonatele curente (X,Y - reale) ale cursorului pe ecran. Ecranul monitorului are 256x176 pixeli şi cursorul poate fi poziţionat în oricare punct al acestei suprafeţe; dacă se încearcă scoaterea cursorului din limitele ecranului, va fi afişat mesajul 'Out of limits', iar programul va fi

oprit cu un mesaj 'HALT'. La începutul programului XCOR şi YCOR sînt nedefinite, ele sînt iniţializate doar prin apelarea procedurii TURTLE cu valorile 127, respectiv 87, coordonatele mijlocului ecranului, (altfel spus "BROASCA a fost palasată în mijlocul bălţii sale").

PENSTATUS

Este o variabilă de tip întreg care păstrează starea curentă a "peniţei" (adică a urmei lăsate de cursor). Poate lua valorile 0 sau 1. Valoarea 0 înseamnă "peniţa jos" - deci cursorul lasă urme în timpul deplasării; 1 înseamnă "peniţa sus" - deci cursorul nu lasă urme.

7.3. PROCEDURILE PACHETULUI TURTLE

In acest paragraf prezentăm textele sursă ale procedurilor pachetului iniţial şi efectul apelării lor precum şi cîteva aplicaţii.

7.3.1. Procedura SPOUT

```
PROCEDURE SPOUT(C:CHAR);
BEGIN
  INLINE(#FD,#21,#3A,#5C,#D0,#7E,#02)
END;
```

Această procedură transmite parametrul C de tip caracter direct prin rutina RST #10 din ROM, evitînd interpretarea eronată de către HP4TM a parametrilor de ieşire ai procedurilor WRITE şi WRITELN (în special cînd se folosesc caractere de control).

Rutina construită în INLINE este:

```
LD  IY,#5C3A
LD  A,(IX+2)
RST #10
```

7.3.2. Procedura CHECK

Această procedură are rolul de a verifica dacă coordonatele cursorului sînt în interiorul ecranului.

```
PROCEDURE CHECK(X,Y:INTEGER);
BEGIN
  IF (X>255) OR (X<0) OR (Y>175) OR (Y<0)
    THEN BEGIN
           WRITE('Out of limits');
           HALT
         END
END;
```

Evident în multe cazuri dorim ca programul să nu se oprească în cazul ieşirii din ecran, în vederea unor reveniri ale cursorului în ecran. Procedura se poate modifica astfel:

```
PROCEDURE CHECK(X,Y:INTEGER;VAR SW:BOOLEAN);
BEGIN
  SW:=TRUE;
  IF (X>255) OR (X<0) OR (Y>175) OR (Y<0)
   THEN SW:=FALSE
END;
```

unde SW este o variabilă globală testată înainte de a se realiza punerea unui punct pe ecran.

7.3.3. Procedura PLOT

```
PROCEDURE PLOT(ON:BOOLEAN;X,Y:INTEGER);
BEGIN
  IF ON
    THEN WRITE(CHR(21),CHR(0))
    ELSE WRITE(CHR(21),CHR(1));
  CHECK(X,Y);
  INLINE(#FD,#21,#3A,#5C,#DD,#46,#02,#DD,#4E,#04,#CD,
         #E5,#22)
END;
```

Dacă ON este TRUE atunci punctul de coordonate (X,Y) va fi plasat, oricare ar fi starea pixelului din acea poziţie. Dacă ON este FALSE, atunci starea pixelului din poziţia (X,Y) se inversează.

Rutina conţinută în INLINE este:

```
LD   IY,#5C3A
LD   B,(IX+2)
LD   C,(IX+4)
CALL #22E5       rutina PLOT din ROM
```

7.3.4. Procedura LINE

Are rolul de a trasa o dreaptă de la poziţia curentă (XCOR,YCOR) la o nouă poziţie (X+XCOR, Y+YCOR). Linia se trasează dacă ON este TRUE şi se inversează starea pixelilor de pe linie dacă ON este FALSE.

```
PROCEDURE LINE(ON:BOOLEAN;X,Y:INTEGER);
VAR SGNX,SGNY:INTEGER;
BEGIN
  CHECK(ROUND(X+XCOR),ROUND(Y+YCOR));
  IF ON
    THEN WRITE(CHR(21),CHR(0))
    ELSE WRITE(CHR(21),CHR(1));
  IF X<0
    THEN SGNX:=-1
    ELSE SGNX:=1;
  IF Y<0
    THEN SGNY:=-1
    ELSE SGNY:=1;
  LINE1(ABS(X),ABS(Y),SGNX,SGNY)
END;
```

Se observă faptul că această procedură apelează la rîndul ei o altă procedură, LINE1, care nu face altceva decît să apeleze rutina DRAW din ROM.

```
PROCEDURE LINE1(X,Y,SX,XY:INTEGER);
BEGIN
  INLINE(#FD,#21,#3A,#5C,#DD,#56,#02,#DD,#5E,#04,
         #DD,#46,#06,#DD,#4E,#08,#CD,#BA,#24)
END;
```

adică:

```
LD   IY,#5C3A
LD   D,(IX+2)
LD   E,(IX+4)
LD   B,(IX+6)
LD   C,(IX+8)
CALL #24BA       rutina DRAW din ROM
```

7.3.5. Procedura INK

```
PROCEDURE INK(C:INTEGER);
BEGIN
  IF (C>=0) AND (C<8)
    THEN
      SPOUT(CHR(16));
  SPOUT(CHR(C))
END;
```

C este un parametru întreg cuprins între 0 şi 7; procedura stabileşte culoarea urmei lăsate de cursor (a "cernelii").

7.3.6. Procedura PAPER

```
PROCEDURE PAPER(C:INTEGER);
BEGIN
IF (C>=0) AND (C<8)
  THEN
    INLINE(1,0,3,#21,0,#58,#DD,#7E,2,7,7,7,#5F,#7E,#E6,
           #C7,#B3,#77,#23,#0B,#78,#B1,#20,#FE);
  SPOUT(CHR(17));
  SPOUT(CHR(8))
END;
```

Procedura stabileşte culoarea fondului ecranului, "hîrtiei", conform culorii asociate parametrului C, un întreg între 0 şi 7 inclusiv.

Rutina din INLINE este:

```
       LD  BC,#0300
       LD  HL,#5800
CICLU  LD   A,(IX+2)
       RLCA
       RLCA
       RLCA
       LD  E,A
       LD  A,(HL)
       AND  #C
       OR   E
       LD  (HL),A
       INC  HL
       DEC  BC
       LD   A,B
       OR   C
       JR   NZ,CICLU
```

7.3.7. Procedura PENDOWN

Procedura modifică starea cursorului, astfel încît el să lase o urmă avînd culoarea asociată parametrului C, care este un întreg între 0 şi 7 inclusiv. Procedura iniţializează cu 0 variabila PENSTATUS:

```
PROCEDURE PENDOWN(C:INTEGER);
BEGIN
  PENSTATUS:=0;
  INK(C)
END;
```

7.3.8. Procedura PENUP

După apelarea procedurii, cursorul nu va mai lăsa urmă. Este utilă pentru a face deplasări dintr-o zonă în alta a ecranului. Procedura inițializează cu 1 variabila PENSTATUS.

```
PROCEDURE PENUP;
BEGIN
  PENSTATUS:=1
END;
```

7.3.9. Proceduri pentru fixare și mișcare cursor

A. Procedura SETHD

```
PROCEDURE SETHD(A:REAL);
BEGIN
  HEADING:=A
END;
```

A este un parametru real care este atribuit variabilei globale HEADING, stabilind astfel orientarea cursorului, pentru care se pot da următoarele valori:

```
  0 - EST  - dreapta
 90 - NORD - sus
180 - VEST - stînga
270 - SUD  - jos
```

B. Procedura SETXY

```
PROCEDURE SETXY(X,Y:REAL);
BEGIN
  XCOR:=X;
  YCOR:=Y
END;
```

Procedura stabilește poziția absolută a cursorului pe ecran în punctul de coordonate (X,Y). Remarcăm că nu se verifică dacă punctul aparține sau nu ecranului. Această verificare se face în procedura LINE, așa cum s-a arătat în subparagraful 7.3.4.

C. Procedura FWD

```
PROCEDURE FWD(L:REAL);
VAR NEWX,NEWY:REAL;
BEGIN
  PLOT(ROUND(XCOR),ROUND(YCOR));
  NEWX:=XCOR+L*COS(HEADING*3.1415926/180);
  NEWY:=YCOR+L*SIN(HEADING*3.1415926/180);
  LINE(TRUE,ROUND(NEWX)-ROUND(XCOR),ROUND(NEWY)-
       ROUND(YCOR));
  XCOR:=NEWX;
  YCOR:=NEWY
END;
```

Procedura deplasează cursorul înainte (FORWARD), cu L unități, în direcția în care este orientată (prin HEADING). O unitate corespunde unui pixel grafic, rotunjind, dacă este cazul, prin adaos sau prin lipsă, noile coordonate.

D. Procedura BACK

Deplasează cursorul cu L unităţi în direcţia opusă orientării ei curente. Orientarea cursorului rămîne neschimbată.

```
PROCEDURE BACK(L:INTEGER);
BEGIN
  FWD(-L)
END;
```

E. Procedura TURN

Procedura are rolul de a modifica orientarea cursorului cu A grade, fără a-l deplasa. Orientarea este mărită în sens trigonometric, invers acelor de ceasornic.

```
PROCEDURE TURN(A:REAL);
BEGIN
     HEADING:=HEADING+A
END;
```

F. Procedura VECTOR

Procedura fixează orientarea, apoi deplasează cursorul cu L unităţi în direcţia dată de A. După deplasare, cursorul rămîne cu aceeaşi orientare.

```
PROCEDURE VECTOR(A,L:REAL);
BEGIN
  SETHD(A);
  FWD(L)
END;
```

G. Procedurile RIGHT şi LEFT

Procedura RIGHT este folosită ca alternativă a procedurii TURN, modificînd orientarea cursorului, în sensul acelor de ceasornic, cu A grade. Procedura LEFT are acelaşi efect, sensul fiind invers.

```
PROCEDURE RIGHT(A:REAL);
BEGIN
  TURN(-A)
END;
```

```
PROCEDURE LEFT(A:REAL);
BEGIN
  TURN(A)
END;
```

H. Procedura ARCR

Această procedură realizează deplasarea cursorului pe un arc de cerc cu lungimea R. Lungimea arcului este determinată de A, unghiul la centru în sens orar. In mod normal R=0,5.

```
PROCEDURE ARCR(R:REAL;A:INTEGER);
VAR I:INTEGER;
BEGIN
  FOR I:=1 TO A DO
    BEGIN
      FWD(R);
      TURN(1)
    END
END;
```

I. Procedura TURTLE

Procedura are rolul de a inițializa starea cursorului, plasîndu-l în mijlocul ecranului, orientat spre EST (dreapta). Fondul este albastru, iar urma cursorului galbenă. Această procedură trebuie apelată la începutul programului deoarece starea cursorului nu este definită în momentul lansării.

```
PROCEDURE TURTLE;
BEGIN
  PAGE;
  SETXY(127,87);
  SETHD(0);
  PAPER(1);
  PENDOWN(6)
END;
```

Observație:

Setul inițial TURTLE mai conține o procedură numită COPY care realizează o copie grafică a ecranului curent la o imprimantă ZX PRINTER. Sugerăm înlocuirea procedurii respective cu o procedură proprie, specifică imprimantei și calculatorului avute la dispoziție.

7.3.10. Proceduri pentru sunet

In pachetul TURTLE există două proceduri pentru generarea sunetelor.

```
PROCEDURE BEEP(FREQUENCY:INTEGER;LENGTH:REAL);
VAR I:INTEGER;
BEGIN
  IF FREQUENCY=0
    THEN
      FOR I:=0 TO ENTIER(12000*LENGTH) DO
    ELSE
      BEEPER(ENTIER(FREQUENCY*LENGTH),
             ENTIER(437500/FREQUENCY-30.128));
  FOR I:=1 TO 100 DO
END;

PROCEDURE BEEPER(A,B:INTEGER);
BEGIN
  INLINE(#DD,#6E,2,#DD,#66,3,#DD,#5E,4,#DD,#56,5,#CD
        #B5,3,#F3)
END;
```

adică:

```
LD   L,(IX+2)
LD   H,(IX+3)
LD   E,(IX+4)
LD   D,(IX+5)
CALL #03B5       rutina BEEPER din ROM
DI
```

7.3.11. Observații și exemple

Bineînțeles, procedurile prezentate anterior pot fi modificate, îmbunătățite, pentru a oferi utilizatorului un cîmp cît mai larg de acțiune.

Cîteva exemple de astfel de modificări:

In programul **DRAW** (P.VII.1) a fost introdusă procedura OVER şi au fost modificate corespunzător procedurile PLOT şi LINE.

```
AEDB   10 PROGRAM DRAW;
AEDB   20 VAR ON:BOOLEAN;
AEE4   30
AEE4   40 PROCEDURE OVER(UNU:INTEGER);
AEE7   50 BEGIN
AEFF   60  WRITE(CHR(21),CHR(UNU))
AF10   70 END;
AF17   80
AF17   90 PROCEDURE PLOT(ON:BOOLEAN;X,Y:INTEGER);
AF1A  100 BEGIN
AF32  110  IF ON THEN OVER(0)
AF3D  120        ELSE OVER(1);
AF56  130  INLINE(#FD,#21,#3A,#5C,#DD,#46,2,#DD,#4E,4,#CD,#E5,#22)
AF63  140 END;
AF6D  150
AF6D  160 PROCEDURE LINE1(X,Y,SX,SY:INTEGER);
AF70  170 BEGIN
AF88  180  INLINE(#FD,#21,#3A,#5C,#DD,#56,2,#DD,#5E,4,#DD,#46);
AF94  185  INLINE(6,#DD,#4E,8,#CD,#BA,#24)
AF9B  190 END;
AFA5  200
AFA5  210 PROCEDURE LINE(ON:BOOLEAN;X,Y:INTEGER);
AFA8  220 VAR SGNX,SGNY:INTEGER;
AFA8  230 BEGIN
AFC0  240  IF ON THEN OVER(0)
AFCB  250        ELSE OVER(1);
AFE4  260  IF X<0 THEN SGNX:=-1 ELSE SGNX:=1;
B013  270  IF Y<0 THEN SGNY:=-1 ELSE SGNY:=1;
B042  280  LINE1(ABS(X),ABS(Y),SGNX,SGNY)
B064  290 END;
B079  300
B079  310 BEGIN
B082  320 PAGE;
B087  330  ON:=TRUE;
B08C  340  PLOT(ON,0,0);
B09E  350  LINE(ON,255,0);
B0B0  360  LINE(ON,0,175);
B0C2  370  LINE(ON,-255,0);
B0D7  380  LINE(ON,0,-175);
B0EC  390  LINE(ON,255,175);
B0FE  395  REPEAT UNTIL INCH<>CHR(0)
B109  400 END {$P}.
```

P.VII.1

In programul **AT** (P.VII.2) a fost introdusă procedura PRINTAT, (din acest moment putîndu-se lucra normal ca şi în BASIC).

```
ACFD   10 PROGRAM AT;
ACFD   20 VAR L,C:INTEGER;
AD06   30
AD06   40 PROCEDURE SPOUT(C:CHAR);
AD09   50 BEGIN
AD21   60  INLINE(#FD,#21,#3A,#5C,#DD,#7E,2,#D7)
AD29   70 END;
AD30   80
AD30   90 PROCEDURE PRINTAT(L,C:INTEGER);
AD33  100 BEGIN
AD4B  110  SPOUT(CHR(22));SPOUT(CHR(L));SPOUT(CHR(C))
AD75  120 END;
AD88  130
AD88  140 BEGIN
AD91  150  READ(L,C);
AD9D  160  PAGE;
ADA2  170  PRINTAT(L,C);
ADAF  180  WRITE(' T E S T ');
ADC3  190 END {$P}.
```

P.VII.2

Pentru trasarea unei elipse propunem următoarea procedură:

```
PROCEDURE ELIPSOID(X,Y,R1,R2:INTEGER);
VAR NEWX,NEWY,I:INTEGER;
BEGIN
  FOR I:=0 TO 360 DO
    BEGIN
      NEWX:=+ROUND(R1*SIN(I));
      NEWY:=Y+ROUND(R2*COS(I));
      PLOT(NEWX,NEWY)
    END
END;
```

Pentru trasarea unui cerc se poate apela procedura anterioară cu R1=R2 în felul următor:

```
PROCEDURE CERC(X,Y,R:INTEGER);
BEGIN
  ELIPSOID(X,Y,R,R)
END;
```

Pentru trasarea unui cerc se poate alege şi următoarea variantă:

```
FOR I:=1 TO 9 DO
BEGIN
  ARCR(0.5,360);
  RIGHT(40)
END;
```

Pentru transmiterea simultană a culorilor pentru INK şi PAPER se poate folosi procedura:

```
PROCEDURE PAPERINK(P,I:INTEGER);
BEGIN
  SPOUT(CHR(17));
  SPOUT(CHR(P));
  SPOUT(CHR(16));
  SPOUT(CHR(I))
END;
```

Programul **GRAFICRECURSIV** (P.VII.3) conţine toate rutinele din pachetul TURTLE precum şi o aplicaţie privind trasarea unor curbe definite recursiv. Sugerăm execuţia programului pentru următoarele date (perechi de numere), reprezentînd valorile iniţiale ale parametrilor SIZE şi DIFF:

(40,15); (40,16);(40,17); (40,25); (50,32); (31,7); (31,12); (31,13); (29,6); (29,8); (20,3).

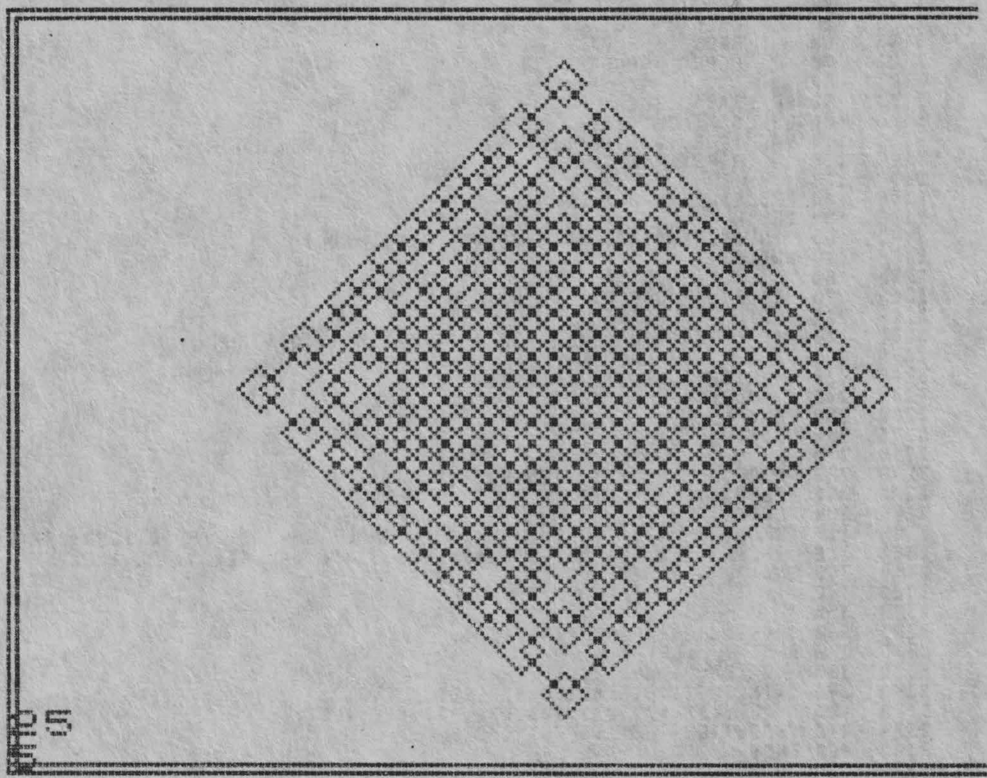
```
B78F    10 PROGRAM GRAFICRECURSIV;
B78F    20 VAR
B798    30  XCOR,YCOR,HEADING:REAL;
B798    40 A,B, PS:INTEGER;
B798    50
B798    60 PROCEDURE SPOUT(C:CHAR);
B79B    70 BEGIN
B7B3    80  INLINE(#FD,#21,#3A,#5C,   #DD,#7E,2,#D7)
B7BB    90 END;
B7C2   100
B7C2   110 PROCEDURE CHECK(X.Y:INTEGER);
B7C5   120 BEGIN
B7DD   130  IF (X>255) OR (X<0) OR    (Y>175) OR (Y<0) THEN
```

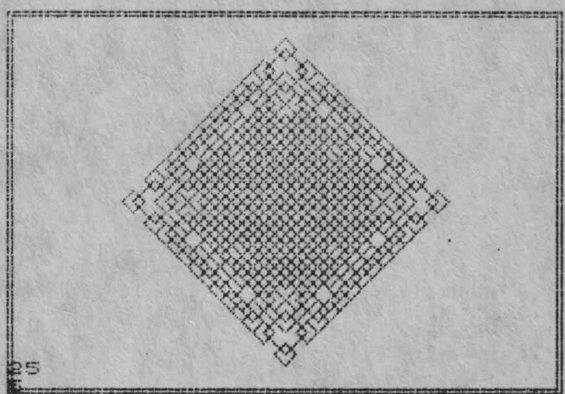
```
 BEGIN
  WRITE('Out of limits');
  HALT
 END
END;
PROCEDURE PLOT(X,Y:INTEGER);
BEGIN
 CHECK(X,Y);
 SPOUT(CHR(20));SPOUT(CHR(PS));SPOUT(CHR(21));SPOUT(CHR(PS));
 INLINE(#FD,#21,#3A,#5C,   #DD,#46,2,#DD,#4E,4,#CD,  #E5,#22)
END;
PROCEDURE LINE1(X,Y,SX,SY:INTEGER);
BEGIN
 INLINE(#FD,#21,#3A,#5C,   #DD,#56,2,#DD,#5E,4,#DD,  #46,6,#DD,#4E,8,#CD,
 #BA,#24)
END;
PROCEDURE LINE(ON:BOOLEAN;X,Y:INTEGER);
VAR SGNX,SGNY:INTEGER;
BEGIN
 CHECK(ROUND(X+XCOR),       ROUND(Y+YCOR));
 SPOUT(CHR(20));SPOUT(CHR( PS));SPOUT(CHR(21) );SPOUT(CHR(PS));
 IF X<0 THEN SGNX:=-1 ELSE SGNX:=1;
 IF Y<0 THEN SGNY:=-1 ELSE SGNY:=1;
 LINE1(ABS(X),ABS(Y),SGNX, SGNY)
END;
PROCEDURE INK(C:INTEGER);
BEGIN
 IF (C>=0) AND (C<8) THEN
 SPOUT(CHR(16));SPOUT(CHR( C))
END;
PROCEDURE PAPER(C:INTEGER);
BEGIN
 IF (C>=0) AND (C<8) THEN
  INLINE(1,0,3,#21,0,#58,   #DD,#7E,2,7,7,7,#5F,#7E,
  #E6,#C7,#B3,#77,#23,#0B,#78,#B1,#20,#EE);
 SPOUT(CHR(17));SPOUT(CHR( 8));
END;
PROCEDURE COPY;
BEGIN
 INLINE(#FD,#21,#3A,#5C,
        #FD,#CB,#01,#CE,
        #CD,#AC,#0E,#FD,
        #CB,#01,#8E,#F3,
        #C9)
END;
PROCEDURE PENDOWN(C:INTEGER);
BEGIN
 PS:=0;
 INK(C)
END;
PROCEDURE PENUP;
BEGIN
 PS:=1
END;
PROCEDURE SETHD(A:REAL);
BEGIN
 HEADING:=A
END;
PROCEDURE SETXY(X,Y:REAL);
BEGIN
 XCOR:=X;
 YCOR:=Y
END;
PROCEDURE FWD(L:REAL);
VAR NEWX,NEWY:REAL;
BEGIN
 PLOT(ROUND(XCOR),ROUND    (YCOR));
 NEWX:=XCOR+L*COS(HEADING* 3.1415926/180);
 NEWY:=YCOR+L*SIN(HEADING* 3.1415926/180);
 LINE(TRUE,ROUND(NEWX)      -ROUND(XCOR),ROUND(NEWY)- ROUND(YCOR));
```

```
BD69   XCOR:=NEWX;
BD7C   YCOR:=NEWY
BD7C  END;
BD9E
BD9E  PROCEDURE BACK(L:REAL);
BDA1  BEGIN
BDB9   FWD(-L)
BDCF  END;
BDE2
BDE2  PROCEDURE TURN(A:REAL);
BDE5  BEGIN
BDFD   HEADING:=HEADING+A;
BE1C  END;
BE26
BE26  PROCEDURE VECTOR(A,L:REAL);
BE29  BEGIN
BE41   SETHD(A);
BE58   FWD(L)
BE66  END;
BE79
BE79  PROCEDURE RIGHT(A:REAL);
BE7C  BEGIN
BE94   TURN(-A)
BEAA  END;
BEBD
BEBD  PROCEDURE LEFT(A:REAL);
BEC0  BEGIN
BED8   TURN(A)
BEE6  END;
BEF9
BEF9  PROCEDURE ARCR(R:REAL; A:INTEGER);
BEFC  VAR I:INTEGER;
BEFC  BEGIN
BF14   FOR I:=1 TO A DO
BF3E   BEGIN
BF41    FWD(R); TURN(1)
BF60   END
BF69  END;
BF78
BF78  PROCEDURE TURTLE;
BF7B  BEGIN
BF93   PAGE;
BF98   SETXY(127,87);
BFB1   SETHD(0);
BFC2   PAPER(1);
BFCF   PENDOWN(6)
BFD3  END;
BFE2  PROCEDURE EXIT;
BFE5  VAR A:INTEGER;
BFE5  BEGIN
BFFD  READ (A);
C006  IF A=1 THEN INLINE (#FD,#21,#3A,#5C,#C3,#3A,#5B) END;
C029
C029  PROCEDURE LINEX(X,Y:INTEGER); BEGIN
C044  LINE(TRUE,X,Y);
C05F  SETXY(X+XCOR,Y+YCOR) END;
C0A0
C0A0   PROCEDURE DIAMANT(X,Y,SIZE,DIFF:INTEGER);
C0A3  BEGIN
C0BB   PLOT (X,Y-SIZE);
C0E1   SETXY(X,Y-SIZE);
C112   LINEX(-SIZE,SIZE);
C12C   LINEX(SIZE,SIZE);
C143   LINEX(SIZE,-SIZE);
C15D   LINEX(-SIZE,-SIZE);
C17A   IF SIZE>4 THEN BEGIN
C190   DIAMANT(X,Y+SIZE,SIZE-DIFF,DIFF);
C1D2   DIAMANT(X,Y-SIZE,SIZE-DIFF,DIFF);
C215   DIAMANT(X-SIZE,Y,SIZE-DIFF,DIFF);
C258   DIAMANT(X+SIZE,Y,SIZE-DIFF,DIFF) END
C29A  END;

C2A4
C2A4  BEGIN
C2AD  TURTLE;
C2B2  PAPER(0);PAGE;
C2C0  REPEAT
C2C0  PAGE;
C2C8  READ(A,B);
C2D4  SPOUT(CHR(22));SPOUT(CHR(0));SPOUT(CHR(0));
```

```
C2F5 DIAMANT(128,88,A,B);
C30A REPEAT UNTIL INCH<>CHR(0);
C31F FOR A:=1 TO 300 DO B:=B+1;
C346  TOUT('DIAMANT ',16384,6912)
C363 UNTIL FALSE
C365 END {$P}.
```



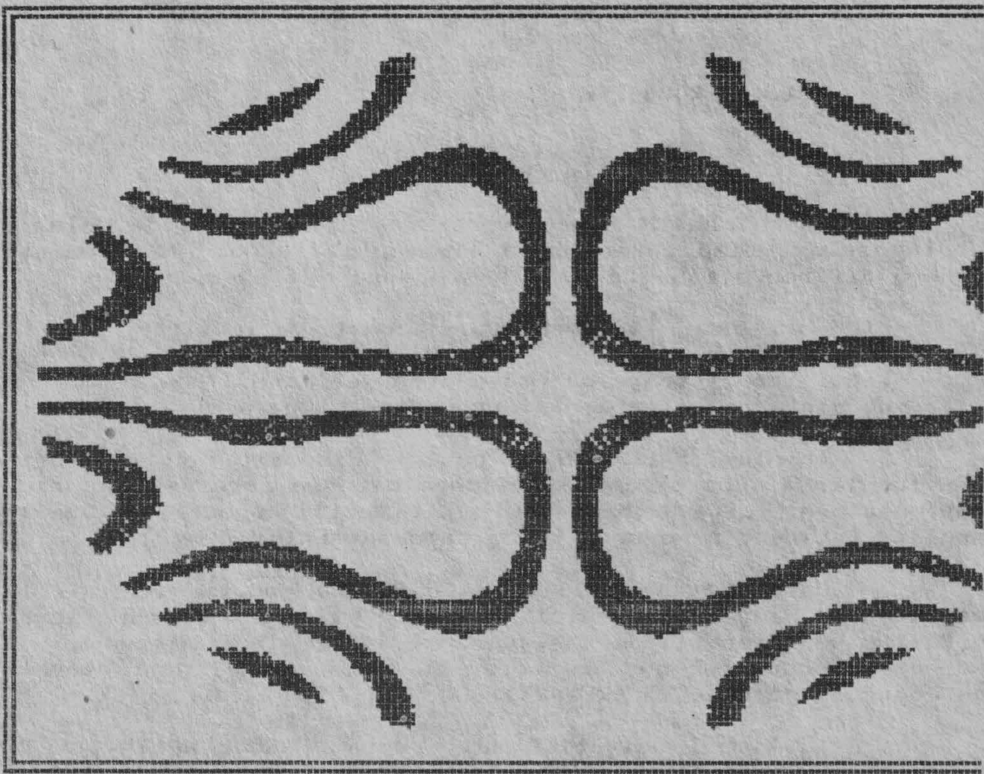


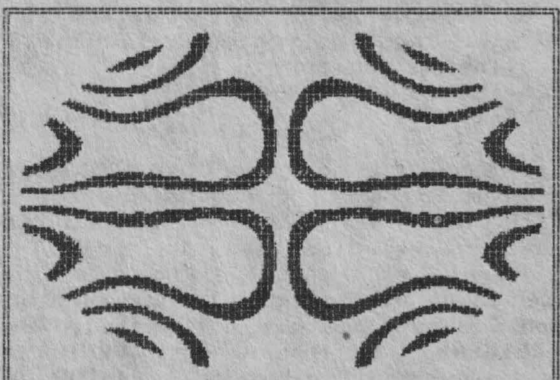
P. VII. 3

Programul ELIPSA (P.VII.4) utilizează doar cîteva din procedurile grafice, realizînd un desen bazat pe trasarea unor elipse din care unele porţiuni sînt ascunse.

```
B1D3
B1D3  { UTILIZARE RUTINE GRAFICE
B1D3      TURTLE- initializare
B1D3      SPOUT - RST  #10
B1D3      CHECK - out of screen$
B1D3      PLOT  - PLOT
B1D3      PAPER - PAPER        }
B1D3
B1D3  PROGRAM ELIPSA;
B1D3  CONST PI=3.14159;
B1D3  VAR XCOR,YCOR,HEADING,RX,RY,X,Y,I,R1,R2:REAL;
B1DC      R,PS:INTEGER;
B1DC
B1DC  PROCEDURE SPOUT(C:CHAR);
B1DF  BEGIN
B1F7   INLINE(#FD,#21,#3A,#5C,#DD,#7E,2,#D7)
B1FF  END;
B206
B206  PROCEDURE CHECK(X,Y:INTEGER);
B209  BEGIN
B221   IF (X>255) OR (X<0) OR (Y>175) OR (Y<0) THEN
B278    BEGIN
B278     WRITE('Out of limits');
B290     HALT
B290    END
B293  END;
B29D
B29D  PROCEDURE PLOT(X,Y:INTEGER);
B2A0  BEGIN
B2B8   CHECK(X,Y);
B2CF   SPOUT(CHR(20));SPOUT(CHR(PS));SPOUT(CHR(21));SPOUT(CHR(PS));
B30B   INLINE(#FD,#21,#3A,#5C,#DD,#46,2,#DD,#4E,4,#CD,#E5,#22)
B318  END;
B322
B322
B322  PROCEDURE PAPER(C:INTEGER);
B325  BEGIN
B33D   IF (C>=0) AND (C<8) THEN
B36C    INLINE(1,0,3,#21,0,#58,#DD,#7E,2,7,7,7,#5F,#7E,#E6,
B37B    #C7,#B3,#77,#23,#0B,#78,#B1,#20,#EE);
B384   SPOUT(CHR(17));SPOUT(CHR(8));
B3A2  END;
B3A9
B3A9
B3A9  PROCEDURE TURTLE;
B3AC  BEGIN
B3C4   PAGE;
B3C9   PAPER(1);
B3D6  END;
B3DC
B3DC  BEGIN
B3E5   TURTLE;
B3EA   RX:=120;RY:=80;
B404   REPEAT
B404   I:=0;
B414   WHILE I<(PI/2-1/RX) DO
B45B    BEGIN
B45B     I:=I+1/RX;
B480     X:=RX*COS(I);
B49D     Y:=RY*SIN(I);
B4BA     R1:=(SQRT(X)*SIN(Y/16))+SQRT(Y)*SIN(X/16)/2;
B519     R:=ROUND(1+COS(R1));
B534     IF R=2 THEN R:=1;
B54C     PS:=R;
B552     PLOT(128+ROUND(X),87+ROUND(Y));
B583     PLOT(128+ROUND(X),87-ROUND(Y));
B5B5     PLOT(128-ROUND(X),87+ROUND(Y));
B5E7     PLOT(128-ROUND(X),87-ROUND(Y))
B615    END;
B61D    RX:=RX-1;RY:=RY-0.6;
B657   UNTIL RX=0;
B678   SPOUT(CHR(22));SPOUT(CHR(0));SPOUT(CHR(0));
B699   REPEAT UNTIL INCH<>CHR (0);
B6AE   FOR R:=0 TO 255 DO
B6C8    BEGIN
B6CB     PLOT(R,0);
```

```
B6D8       PLOT(255-R,175)
B6EC      END;
B6F4    FOR R:=0 TO 175 DO
B70E      BEGIN
B711       PLOT(0,R);
B71E       PLOT(255,175-R)
B732      END;
B73A    TOUT('elipsa   ',16384,6912)
B757  END {$P}.
```





P.VII.4

VIII. MIC MANUAL DE OPERARE PASCAL HP4TM

8.1. INTRODUCERE

8.1.1. Punerea în funcţiune

Pascal HP4TM este o versiune rapidă, uşor de folosit şi în acelaşi timp puternică a limbajului Pascal. Ea prezintă cîteva diferenţe faţă de PASCAL standard, de exemplu:

- -FILE nu este implementată, totuşi variabilele pot fi stocate pe bandă;
- -tipul înregistrare nu poate avea variante;
- -nu sînt admişi parametri procedură şi parametri funcţie.

In plus, PASCAL HP4TM pune la dispoziţia utilizatorilor, pe lîngă subprogramele standard care se regăsesc în orice implementare, cîteva proceduri şi funcţii specifice, unele apelînd rutine din memoria ROM a calculatorului.

Pascal HP4TM utilizează diferite coduri de control, mai ales în cadrul editorului. Desigur, calculatoarele pot avea diferite configuraţii de tastatură şi astfel diferite moduri de formare a codurilor de caractere. In acest manual, caracterele de control vor fi <CR> (RETURN), CC, CH, CI, CP, CS şi CX.

In timpul editării unui program PASCAL HP4TM pot fi folosite următoarele caractere de control:

- -<CR> (RETURN, ENTER)- se foloseşte pentru a termina linia;
- -CC (CAPS SHIFT+1)-se revine în editor;
- -CH (CAPS SHIFT+0)-şterge ultimul caracter introdus;
- -CI (CAPS SHIFT+8)-deplasează la următoarea poziţie TAB;
- -CP (CAPS SHIFT+3)-schimbă ieşirea pe printer (dacă există), iar dacă ieşirea era pe printer, se revine pe monitor;
- -CX (CAPS SHIFT+5)-şterge întreaga linie;
- -CS (CAPS SHIFT+SPACE)-BREAK.

Incărcarea HP4TM se face cu comanda LOAD "". Mai întîi se încarcă un scurt program în BASIC, cu autostart, care va încărca efectiv blocul HP4TM. După ce HP4TM s-a încărcat, el va fi lansat automat în execuţie şi va afişa mesajul:

Top of RAM?

Trebuie să se introducă un număr zecimal pozitiv pînă la 65535 (acest număr reprezentînd noua valoare a variabilei de sistem RAMTOP), apoi <CR> (RETURN), fie direct <CR> (RETURN) (în acest caz variabila de sistem nemodificîndu-se). RAMTOP se află memorat pe doi octeţi la adresa 23730.

Stiva compilatorului este plasată aici, deci rezervarea unei zone de memorie pentru rutine proprii în cod

maşină sau pentru memorarea altor informaţii se poate face prin tastarea unei valori mai mici decît RAMTOP-ul real. In versiunea pentru ZX SPECTRUM, RAMTOP-ul real este considerat a fi începutul zonei de caractere grafice definite de utilizator (UDG).

Se cere apoi: Top of RAM for 'T'.

Se poate introduce fie un număr zecimal, fie rămîne implicit valoarea anterioară pentru 'Top of RAM'. Valoarea introdusă va fi considerată adresa de început a stivei pentru cazul cînd se execută codul obiect rezultat prin comanda 'T' a editorului. Este necesar să se definească o stivă de rulare diferită de RAMTOP-ul real dacă, de exemplu, s-au scris extensii la modulele de execuţie care se vor introduce în locaţiile de deasupra RAMTOP-ului.

In fine, se cere: Table size?

Ceea ce se introduce va fi mărimea zonei de memorie care trebuie alocată tabelului de simboluri al compilatorului. Ca mai înainte, se poate introduce un număr zecimal urmat de <CR>, sau numai <CR>, în care caz pentru mărimea acestei zone de memorie va fi adoptată o valoare implicită (memoria RAM disponibilă împărţită la 16). In aproape toate cazurile, valoarea implicită asigură spaţiu mai mult decît suficient pentru tabelul de simboluri. Tabelul de simboluri nu se poate extinde peste adresa #8000=32768 în zecimal. Dacă se menţionează că va avea loc această depăşire, se va cere din nou să se introducă 'Top of RAM' şi celelalte opţiuni.

Opţional se poate include E înainte de numărul introdus, dacă se doreşte ca editorul intern să nu fie reţinut pentru a fi utilizat cu compilatorul, de exemplu dacă se foloseşte în acest scop un editor propriu.

In acest stadiu, compilatorul şi editorul integral (dacă a fost reţinut) vor fi relocatate după tabelul de simboluri, iar execuţia transferată editorului.

8.1.2. Compilarea şi lansarea în execuţie

Odată ce a fost invocat, compilatorul generează un listing de forma:

    xxxx nnnn textul liniei sursă

unde:

- xxxx este adresa la care începe codul generat de această linie,
- nnnn este numărul liniei, cu zerourile de aliniere suprimate.

Dacă o linie conţine mai mult de 80 de caractere, atunci compilatorul va introduce caractere 'linie nouă' <CR>, în aşa fel încît lungimea unei linii să nu depăşească 80 de caractere.

Listarea poate fi dirijată către printer prin folosirea opţiunii P, dacă aceasta este acceptată.

Listarea poate fi oprită în orice moment, apăsînd CS (BREAK); în continuare, cu CC se revine în editor, iar cu orice altă tastă se continuă listarea.

Dacă pe parcursul compilării este detectată o eroare, atunci va fi afişat mesajul '*ERROR*', urmat de caracterul '^', plasat după simbolul care a generat eroarea şi de un număr de eroare. Listarea se va opri; apăsînd E, linia respectivă este adusă în zona de editare pentru efectuarea corecţiilor, iar apăsînd P în linia de editare este adusă linia anterioară celei care a produs eroarea.

8.2. EDITORUL

8.2.1. Introducere în editor

Editorul furnizat cu toate versiunile Pascal 4 Hisoft este un editor simplu, la nivel de linie, uşor de exploatat şi care editează programele rapid şi eficient.

Editorul este lansat automat în execuţie după ce HP4TM este încărcat de pe casetă, afişînd mai întîi mesajul:

Copyright Hisoft 1983,84
All rights reserved

şi apoi prompterul editorului: '>'. Din acest moment se pot introduce linii de comandă cu formatul următor:

c n1, n2, s1, s2 <CR>

unde:

c - este comanda care urmează a fi executată;
n1, n2- sînt numere în intervalul [1,32767] inclusiv;
s1, s2- sînt şiruri cu maximum 20 de caractere.

Virgula este utilizată pentru a separa diferitele argumente (dar acest separator poate fi schimbat, vezi comanda S), iar spaţiile sînt ignorate, cu excepţia celor din şirurile de caractere.

Nici unul dintre aceste argumente nu este obligatoriu, deşi unele comenzi (cum ar fi comanda **D**) nu acţionează pînă nu sînt specificate **n1** şi **n2**. Editorul memorează numerele şi şirurile introduse chiar dacă nu se specifică unul dintre argumentele liniei de comandă; va utiliza aceste valori introduse anterior, acolo unde este cazul.

Iniţial **n1** şi **n2** au valoarea 10, iar şirurile sînt vide. Dacă se introduce o linie de comandă incorectă, de exemplu F-1, 100, HELLO, ea va fi ignorată şi se va afişa mesajul 'Pardon?'. Linia trebuie introdusă corect, adică F1, 100, HELLO. Acelaşi mesaj de eroare apare şi dacă lungimea lui **s2** depăşeşte 20; dacă lungimea lui **s1** depăşeşte 20, caracterele în plus vor fi ignorate.

Comenzile pot fi introduse cu litere mari sau mici.

La introducerea unei linii de comandă pot fi folosite toate funcţiile de control descrise anterior, (de exemplu CX pentru a şterge toată linia).

Paragraful următor detaliază diferitele comenzi disponibile în editor.

Observaţie:

-dacă unele argumente sînt încadrate între simbolurile '<' şi '>', atunci respectivele argumente trebuie să fie prezente, altfel comanda nu acţionează.

8.2.2. Comenzile editorului

a) Inserarea textului

Textul poate fi introdus într-o filă de text, fie introducînd un număr de linie, un spaţiu şi apoi linia program, fie utilizînd comanda I. La introducerea unui text pot fi utilizate şi funcţiile de control CX, CI, CC, CP şi DELETE.

Comanda I n,m

Cu ajutorul acestei comenzi se obţine intrarea în modul automat de inserare: vor fi afişate numerele de linie, începînd de la **n**, cu pasul **m**. După afişarea numărului de linie se introduce linia program. Aici se pot folosi funcţiile de control. Linia de text se termină cu <CR>. Pentru a ieşi din acest mod se foloseşte funcţia de control CC.

Dacă se introduce o linie a cărui număr există în text, atunci linia existentă va fi înlocuită cu cea nouă după tastarea lui <CR>. Dacă incrementarea automată a numărului de linie produce un număr de linie mai mare decît 32767, se va ieşi automat din modul inserare.

b) Listarea textului

Textul poate fi inspectat folosind comanda L; numărul de linii afişat deodată la executarea acestei comenzi este fixat iniţial, dar poate fi modificat cu comanda K.

Comanda L n,m

Această comandă listează textul curent pe dispozitivul de afişare, de la linia cu numărul **n**, pînă la linia cu numărul **m** inclusiv. Valoarea implicită a lui **n** este întotdeauna 1, iar a lui **m** este 32767, adică valorile implicite nu sînt luate din argumentele introduse anterior. Pentru a lista întregul fişier de text, se utilizează simplu L, fără alte argumente. După listarea unui anumit număr de linii (fixat prin comanda K), listarea se va opri; cu funcţia de control CC se revine în editor iar cu oricare altă tastă se continuă listarea.

Comanda K n

K stabileşte numărul liniilor de ecran care urmează să fie listate înainte ca listarea să fie întreruptă, aşa cum este descris la comanda L. Este calculată şi stocată valoarea **n** MOD 256.

c) Editarea textului

In editor există diverse comenzi pentru corectarea, ştergerea, mutarea sau renumerotarea liniilor.

Comanda D <n,m>

Toate liniile de la **n** la **m** inclusiv sînt şterse din fişierul de text. Dacă **m=n** este ştearsă o singură linie.

Comanda M n,m

Această comandă mută linia **n** la linia **m**; vechea linie **m**, dacă exista, se va pierde, iar linia **n** rămîne neafectată. Dacă linia **n** nu există, comanda este inefectivă.

Comanda N <n,m>

Utilizarea comenzii **N** are ca efect renumerotarea liniilor programului sursă, prima linie primind numărul **n**, iar pasul fiind **m**. Dacă renumerotarea duce la numere de linii mai mari decît 32767, atunci va fi păstrată numerotarea iniţială.

Comanda F n,m,f,s

In textul cuprins între liniile n<m este căutat şirul f. Dacă un asemenea şir de caractere este identificat, atunci este afişată linia care îl conţine şi apoi se intră în modul editare. Se pot folosi subcomenzile F, S ale modului editare. Comanda F se termină cu <CR>.

Valorile numerelor de linii şi cele două şiruri puteau fi specificate anterior prin oricare altă comandă şi în acest caz este suficient să se introducă doar F.

Comanda E n

Editează linia cu numărul n. Dacă există, ea este copiată într-o zonă tampon şi afişată pe ecran (împreună cu numărul de linie). Numărul de linie este apoi afişat din nou, imediat dedesubt şi se trece în modul editare. In modul editare sînt disponibile următoarele subcomenzi:

(spaţiu) - deplasează cursorul cu o poziţie spre dreapta, marcînd caracterul următor din linie;

CH - deplasează cursorul cu o poziţie spre stînga, marcînd caracterul anterior;

CI - deplasează cursorul spre dreapta, pînă la următoarea poziţie TAB;

<CR> - termină editarea liniei curente, cu păstrarea tuturor modificărilor;

Q - termină editarea liniei curente fără să reţină nici una din modificările făcute;

R - reîncarcă linia în tamponul editorului, adică ignoră toate modificările făcute şi reface linia originală;

L - listează restul liniei care a mai rămas de corectat, adică partea care urmează după poziţia curentă a cursorului; se rămîne în modul editare, cu cursorul repoziţionat pe începutul liniei;

K - şterge caracterul care se află la poziţia curentă a cursorului;

Z - şterge toate caracterele de la poziţia curentă a cursorului (inclusiv) pînă la sfîrşitul liniei;

F - găseşte apariţia următoare a şirului f de caractere definit anterior printr-o linie de comandă F; această subcomandă va termina automat editarea liniei curente (cu menţinerea modificărilor) dacă aici nu este identificată o altă apariţie a şirului f; dacă o apariţie a şirului f este detectată într-o linie următoare (între limitele specificate anterior), atunci va fi introdus modul editare pentru linia în care a fost găsit şirul f; după o căutare încheiată cu succes, cursorul va fi poziţionat la începutul liniei;

S - înlocuieşte apariţia curentă a şirului f cu şirul s definit într-o linie de comandă F, după care se execută o subcomandă F, adică se caută următoarea apariţie a şirului f;

I - inserează caractere începînd de la poziţia curentă a cursorului; începutul inserării este marcat prin transformarea cursorului în '*'; se rămîne în acest submod pînă cînd se apasă <CR>; atunci se revine în

modul editare, cu cursorul poziţionat după ultimul caracter inserat; folosind DELETE în cadrul acestui submod caracterul de la stînga cursorului va fi şters, iar cu CI se va deplasa cursorul în poziţia următoare, inserînd spaţii;

X - avansează cursorul la sfîrşitul liniei şi se trece în submodul de inserare;

C - permite ca peste caracterul aflat la poziţia curentă a cursorului să se scrie alt caracter; începutul modificării este marcat prin transformarea cursorului în '+'; se rămîne în submodul de modificare pînă cînd se apasă <CR> şi atunci se revine în modul editare cu cursorul poziţionat după ultimul caracter modificat; DELETE în cadrul acestui submod mută cursorul cu o poziţie spre stînga, iar CI nu are nici un efect.

d) Comenzi pentru lucrul cu banda

Comanda P n,m,s

Liniile din domeniul definit prin **n**<**m** sînt salvate pe casetă în format HP4TM, sub denumirea specificată prin şirul s. Aceste argumente pot fi specificate şi printr-o comandă anterioară. In cazul în care se doreşte realizarea mai multor copii, pentru copiile 2,3,..., este suficientă tastarea comenzii P fără argumente. (In timpul salvării filei text în fişierul cu numele **s** va fi afişat mesajul 'Busy...')

Comanda G,,s

Pe bandă este căutat un fişier în format HP4TM, cu denumirea **s**. Vor fi afişate denumirile fişierelor întîlnite în timpul derulării benzii, precedate de mesajul 'Found' Odată găsit fişierul căutat, se va afişa mesajul 'Using', urmat de numele fişierului, iar acesta va fi încărcat în memorie. Dacă pe timpul încărcării este detectată o eroare, va fi afişat mesajul 'Checksum error' sau 'Tape error', iar încărcarea este oprită.

Dacă şirul **s** este vid, va fi încărcat primul fişier HP4TM întîlnit pe bandă, indiferent de denumirea lui.

Căutarea şi încărcarea de pe bandă se poate opri apăsînd CS; apăsînd apoi CC se revine în editare. Dacă există deja o filă text în memorie, fişierul citit de pe bandă va fi adăugat la aceasta şi fila text rezultată va fi renumerotată, începînd de la numărul de linie 1 cu pasul 1.

e) Compilarea şi lansarea în execuţie din editor

Comanda C n

Se compilează textul care începe la linia cu numărul **n**. Dacă nu se specifică un număr de linie, textul va fi compilat de la prima linie existentă.

Comanda R

Codul obiect compilat anterior va fi executat, dar numai dacă sursa nu a fost modificată între timp.

Comanda T n

Textul sursă este compilat de la linia **n** sau de la început dacă **n** este omis, iar dacă compilarea se termină cu succes, apare mesajul 'Ok?'; dacă răspunsul este 'Y', atunci codul obiect produs prin compilare este deplasat la sfîrşitul modulelor de execuţie (distrugînd compilatorul), apoi modulele de execuţie şi codul obiect vor fi salvate pe bandă, denumirea de fişier fiind cea specificată pentru fişierul **f** definit anterior. Ulterior acest fişier se poate încărca în memorie, folosind încărcătorul HP4TM, după care automat va fi executat codul obiect. Intrucît codul obiect este deplasat la sfîrşitul modulelor de execuţie, după comanda **T** compilatorul nu va mai fi în memorie şi va trebui reîncărcat de pe bandă.

Dacă nu se doreşte salvarea pe bandă, răspunsul la întrebarea 'Ok?'va fi orice caracter diferit de 'Y'; controlul va reveni la editor, acesta funcţionînd normal, întrucît codul obiect nu a fost deplasat.

f) Alte comenzi

Comanda B

Această comandă redă controlul sistemului de operare. In cazul calculatorului ZX SPECTRUM, se revine în BASIC. Pentru a reda controlul compilatorului se foloseşte

RANDOMIZE USR 24598,

caz în care textul existent se şterge, fie cu comanda

RANDOMIZE USR 24603

cînd textul Pascal sursă se păstrează.

Comanda O n,m

Se utilizează pentru a codifica un eventual text, obţinut, de exemplu, cu ajutorul unui alt editor de texte. Textul este citit într-o zonă tampon în forma expandată şi apoi dus înapoi în fişier sub formă codificată.

Comanda S,,d

Cu această comandă se poate schimba delimitatorul folosit pentru separarea argumentelor într-o linie de comandă. La pornirea editorului virgula (',') este luată ca delimitator; aceasta poate fi schimbată, folosind comanda S, cu primul caracter din şirul **d** specificat în comandă.

Menţionăm că spaţiul nu poate servi ca separator.

Comanda V

Afişează valorile curente pentru **n1**, **n2**, **s1**, **s2**.

Comanda X

Afişează în hexa adresa de sfîrşit a compilatorului.

Comanda W n,m,s

Funcţionează la fel ca **P** cu excepţia faptului că fila text nu se salvează în format standard HP4TM. Textul cuprins între liniile **m** şi **n** se va salva pe bandă într-un format

special sub numele s. Textul astfel salvat va fi folosit direct la compilare cu ajutorul opţiunii F a compilatorului.

8.3. Opţiuni ale compilatorului

Sintaxa specificării opţiunilor de compilare este dată în diagrama de sintaxă din fig.8.1.

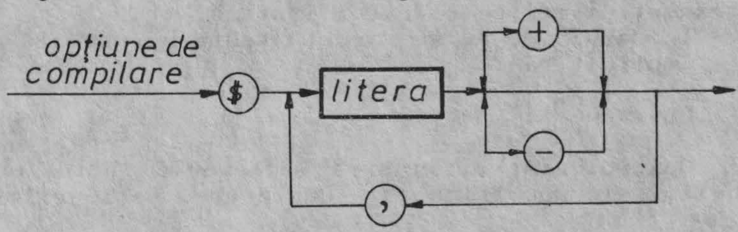


fig.8.1

Observaţie:
-opţiunile compilatorului care admit specificaţia '+', respectiv '-', pot fi anulate prin opţiunea complementară.

Opţiunea L

Controlează listarea programului text şi a adreselor codului maşină generat de compilator.

Cînd se menţionează L+, se listează textul programului.

Pentru L- sînt listate numai liniile în care este detectată o eroare.

Implicit: L+

Opţiunea O

Controlează dacă se fac unele verificări de depăşire. Pentru înmulţirea şi împărţirea numerelor întregi, ca şi pentru toate operaţiile cu numere reale, aceste verificări se fac întotdeauna. In cazul O+ se fac verificări la adunarea şi scăderea numerelor întregi.

In cazul O-, aceste verificări nu se fac.

Implicit: O+.

Opţiunea C

Controlează testarea claviaturii în timpul executării programului în cod maşină. Dacă se specifică C+, atunci execuţia se opreşte cînd se apasă CC, cu apariţia unui mesaj HALT.

Acest control se face înaintea tuturor ciclurilor, procedurilor şi funcţiilor. Utilizatorul poate folosi această facilitate pentru a determina, în cursul unei depanări, ce structuri repetitive nu se termină corect. Această opţiune trebuie, evident, să fie dezactivată dacă vrem ca programul obiect să ruleze rapid.

In cazul C-, verificarea de mai sus nu se face.

Implicit: C+

Opţiunea S

Controlează testarea stivei.

In cazul S+, la apelarea fiecărei proceduri sau

funcţii se face o verificare dacă există probabilitatea ca stiva să fie depăşită în blocul care urmează. Dacă există pericolul ca în timpul execuţiei stiva să se suprapună peste zona afectată variabilelor dinamice sau peste program, execuţia este oprită şi se afişează mesajul 'Out of RAM at PC=XXXX'. Evident, nu este o certitudine; dacă o procedură foloseşte mult stiva, atunci programul se poate într-adevăr 'prăbuşi'. Pe de altă parte, în cazul cînd o funcţie foloseşte puţin stiva, oprirea execuţiei se poate dovedi inutilă.

In cazul S-, nu se execută testarea stivei.

Implicit: S+.

Opţiunea A:

Controlează verificarea încadrării indicilor tablourilor în limitele specificate la declararea tablourilor respective.

Dacă se specifică A+, iar indicele unui element de tablou este prea mare sau prea mic, execuţia programului este oprită, fiind afişat mesajul 'Index too high', sau 'Index too low'.

In cazul A-, această verificare nu se mai face.

Implicit: A+.

Opţiunea I

Folosind aritmetica numerelor întregi pe 16 biţi în cod complementar, are loc o depăşire cînd se execută una din operaţiile <, >, <=, >= dacă diferenţa dintre argumente este mai mare decît MAXINT (32767). In acest caz rezultatul comparării este incorect. In mod normal, aceasta nu supără; dacă, însă, utilizatorul doreşte să compare asemenea numere, I+ va asigura corectitudinea rezultatelor. O situaţie analoagă poate apare în aritmetica numerelor reale, cînd va fi generată o eroare de depăşire dacă argumentele diferă prin mai mult decît aproximativ 3.4E38.

In cazul I- nu se face verificarea rezultatelor comparărilor de mai sus.

Implicit: I-

Opţiunea P

Dacă se foloseşte opţiunea P, dispozitivul la care va fi transmis listingul după compilare va fi comutat, adică dacă înainte era utilizat monitorul, după P va fi utilizat printerul şi invers. Observaţi că această opţiune nu este urmată de '+' sau de '-'.

Implicit: Este utilizat ecranul video.

Opţiunea F

Litera F trebuie să fie urmată de un spaţiu şi apoi denumirea unui fişier, formată din 8 caractere. Dacă denumirea fişierului are mai puţin de 8 caractere, se completează cu spaţii.

Prezenţa acestei opţiuni va determina includerea, la sfîrşitul liniei curente, a textului sursă Pascal din fişierul specificat. Este utilă în cazul cînd programatorul doreşte să alcătuiască pe bandă o 'bibliotecă' din multe proceduri şi funcţii proprii şi apoi să o includă în diferite programe.

Programul trebuie să fi fost salvat cu comanda **W** din

editor. In majoritatea sistemelor trebuie folosită opțiunea L-, altfel viteza de compilare va fi mică.

Exemplu:

{$L-,F MATRIX } include de pe bandă textul fișierului MATRIX

Cînd se editează programe foarte mari, este posibil să nu rămînă loc suficient în memoria calculatorului pentru ca programul sursă și codul obiect să fie prezente simultan. Este posibil, totuși să fie compilate asemenea programe, salvîndu-le pe bandă și folosind opțiunea F. In acest caz, în RAM vor fi - în orice moment - numai 128 octeți din sursă, rămînînd mult loc pentru codul obiect.

Această opțiune nu poate fi inclusă într-o structură repetitivă.

Observație:

Opțiunile compilatorului se pot utiliza și în mod selectiv.

Astfel, secțiunile deja depanate pot fi compactate și făcute să ruleze mai rapid, prin dezactivarea verificărilor inutile și reținînd aceste verificări numai pentru părțile netestate ale codului.

ANEXA 1. CUVINTE REZERVATE SI IDENTIFICATORI PREDEFINITI

A.1.1. Cuvinte rezervate

AND	ARRAY	BEGIN	CASE	CONST	DIV
DO	DOWNTO	ELSE	END	FOR	FORWARD
FUNCTION	GOTO	IF	IN	LABEL	MOD
NIL	NOT	OF	OR	PACKED	PROCEDURE
PROGRAM	RECORD	REPEAT	SET	THEN	TO
TYPE	UNTIL	VAR	WHILE	WITH	

A.1.2. Simboluri speciale

+	-	*	/		
=	<>	<	<=	>=	>
(	)	[	]		
{	}	(*	*)		
^	:=	.	,	;	:
'	..	#	$		

A.1.3. Identificatori predefiniţi

ABS	ADDR	ARCTAN	BOOLEAN	CHAR	CHR
COS	ENTIER	EOLN	EXP	FALSE	FRAC
HALT	INCH	INLINE	INTEGER	INP	LN
MARK	MAXINT	NEW	ODD	ORD	OUT
PAGE	PEEK	POKE	PRED	RANDOM	READ
READLN	REAL	RELEASE	ROUND	SIN	SIZE
SQR	SQRT	SUCC	TAN	TIN	TOUT
TRUE	TRUNC	USER	WRITE	WRITELN	

unde avem:

-constante: MAXINT=32767;

-tipuri: BOOLEAN=(FALSE,TRUE);
CHAR (Setul de caractere ASCII extins);
REAL (Submulţime a numerelor reale)

-proceduri: WRITE; WRITELN; READ; READLN; PAGE; HALT; USER; POKE; INLINE; OUT; NEW; MARK; RELEASE; TIN; TOUT;

-funcţii: ABS; SQR; ODD; RANDOM; ORD; SUCC; PRED; INCH; EOLN; PEEK; CHR; SQRT; ENTIER; ROUND; TRUNC; FRAC; SIN; COS; TAN; ARCTAN; EXP; LN; ADDR; SIZE; INP;

ANEXA 2. HP4TM -COMPILATOR PASCAL ZX SPECTRUM-MEMORATOR

1. COMENZI DE EDITARE

I n, m -inserare de text sursă;
-n=numărul liniei;
-m=pasul de numerotare;
-după fiecare linie se tastează <CR>, în acest moment apărînd numărul liniei următoare;
-ieşirea din înserare se face cu <caps-1>;

L n,m -listare program;
-n=numărul primei linii listate;
-m=numărul ultimei linii listate;

D n,m -ştergere linii;
-n=numărul ultimei linii şterse;

N n,m -renumerotare a liniilor programului;
-n=noua etichetă a primei linii;
-m=noul pas de numerotare;

E n -editarea liniei n;

F x,y,string1,string2
-între liniile x şi y se caută şirul1 şi eventual se înlocuieşte cu şirul2.

2. MEMORAREA, INCARCAREA SI TIPARIREA PROGRAMELOR

P n,m;s -salvare program;
-textul sursă cuprins între liniile n şi m va fi plasat pe casetă sub numele s;

G ,,s -încărcare program de pe casetă;

W n,m,s -salvarea unor porţiuni de program;
-porţiunile astfel salvate pot fi inserate în program folosind opţiunea compilatorului { $F nume program};

{ $F s } -apariţia acestei opţiuni în textul programului are ca efect (în timpul compilării) introducerea în acel punct a unei proceduri salvate cu W sub numele s;

{ $L- } -la întîlnirea acestei opţiuni în timpul compilării se sistează listarea textului sursă pe (rezultă creşterea vitezei de compilare);

B -reîntoarcere în BASIC;
-revenirea în PASCAL se face cu
-RANDOMIZE USR 24603 cu păstrarea textului;
-RANDOMIZE USR 24598 cu ştergerea textului;

{ $P } -se comută canalul de ieşire de la monitor la imprimantă şi invers.

3. ALTE COMENZI

C n -compilare începînd cu linia n;

K n -fixează numărul de linii afişate simultan pe

ecran
M n, m -copierea liniei n în linia m;
R -lansare program (RUN);
T n -compilarea programului şi salvarea codului obiect;
V -indică forma comenzii F n,m,f,s;
X -furnizează adresa de sfîrşit a compilatorului în hexa.

4. OPTIUNI DE EDITARE

space -cursor dreapta;
caps o -cursor stînga;
caps 5 -ştergerea liniei inclusiv eticheta;
caps 8 -salt la următorul TAB;
enter -ieşire din editare cu validarea modificărilor;
C -scrie peste caracterele existente; (ieşire cu <CR>;
F -caută;
I -inserează;
K -şterge caracterul de sub cursor;
L -listarea liniei cu modificările făcute;
Q -ieşire din editare fără validarea modificărilor;
R -reîncărcarea liniei fără modificări;
S -înlocuieşte;
X -salt la sfîrşitul liniei;
Z -şterge toate caracterele de după cursor pînă la sfîrşitul liniei.

5. MESAJE DE EROARE

1 -număr prea mare;
2 -lipseşte ";";
3 -identificator nedeclarat;
4 -lipseşte un identificator;
5 -s-a folosit ":=" în loc de "=" în declararea constantelor;
6 -lipseşte "=";
7 -identificator ce nu poate să apară în membrul stîng al unei atribuiri;
8 -lipseşte ":=";
9 -lipseşte ")";
10 -tip eronat;
11 -lipseşte ".";
12 -lipseşte un factor;
13 -lipseşte o constantă;
14 -acest identificator nu este o constantă;
15 -lipseşte THEN;
16 -lipseşte DO;
17 -lipseşte TO sau DOWNTO;
18 -lipseşte "(";
19 -incompatibilitate de tip;
20 -lipseşte OF;
21 -lipseşte ",";
22 -lipseşte ":";
23 -lipseşte PROGRAM;
24 -lipsă variabilă deoarece parametrul e de tip -variabilă;
25 -lipseşte BEGIN;
26 -lipseşte o variabilă la utilizarea lui READ;
27 -acest tip de expresii nu pot fi comparate;

28 -tipul utilizat trebuie să fie INTEGER sau REAL;
29 -acest tip de variabilă nu poate fi citit cu READ;
30 -acest identificator nu este un tip;
31 -exponentul trebuie să fie un număr real;
32 -lipseşte o expresie scalară (nenumerică);
33 -nu sînt permise şiruri vide; folosiţi CHR(0);
34 -lipseşte [;
35 -lipseşte];
36 -indicii de tablou trebuie să fie de tip scalar;
37 -lipseşte "..";
38 -lipseşte "," sau "]" într-o declaraţie de tablou;
39 -limita inferioară > limita superioară;
40 -mulţime mai mare de 256 dé elemente;
41 -identificatorul de după FUNCTION ( ) trebuie să fie identificator de tip;
42 -lipseşte "," sau "]" într-o mulţime;
43 -lipseşte ".." sau "]" într-o mulţime;
44 -tipul parametrului trebuie să fie un identificator de tip;
45 -a nu se folosi o mulţime vidă ca prim factor în afara unei atribuiri;
46 -lipseşte un tip scalar, inclusiv REAL;
47 -lipseşte un tip scalar, exclusiv REAL;
48 -mulţimile nu sînt compatibile;
49 -"<" şi ">" nu pot să apară în comparaţii de mulţimi;
50 -lipseşte FORWARD,LABEL,CONST,VAR,TYPE,sau BEGIN;
51 -lipseşte un număr hexa;
52 -comanda POKE nu poate fi folosită pentru mulţimi;
53 -matrice prea mare (>64);
54 -lipseşte END sau ";" într-o declaraţie de articol;
55 -lipseşte un identificator de cîmp;
56 -lipseşte variabila de după WITH;
57 -variabila de după WITH trebuie să fie de tip articol;
58 -identificatorul de cîmp nu e precedat de WITH;
59 -lipseşte valoarea întreagă după LABEL;
60 -lipseşte valoarea întreagă după GOTO;
61 -pointer indicînd un cîmp eronat;
62 -pointer nedefinit;
63 -parametrul pentru SIZE trebuie să fie variabilă;
64 -se pot face doar teste de egalitate pentru pointeri;
67 -singura formă de extragere a întregilor cu 2 caractere este e:m:H;
68 -şirurile de caractere nu trebuie să conţină EOL;
69 -parametrii instrucţiunilor NEW, MARK şi RELEASE trebuie să fie variabile de tip pointer;
70 -parametrii instrucţiunii ADDR trebuie să fie variabile.

6. ERORI DE EXECUTIE

1 -oprire;
2 -depăşire;
4 -împărţire la zero;
5 -indice prea mare;
6 -indice prea mic;
7 -eroare într-o rutină matematică;
8 -număr prea mare;
9 -lipseşte un număr;
10 -linie prea lungă;
11 -lipseşte un exponent.

BIBLIOGRAFIE

1. Şerbănaţi, L-D., Iorga, V., Cristea, V., Moldoveanu, F., *Programarea sistematică în limbajele PASCAL şi FORTRAN*, Ed.Tehnică, Bucureşti, 1984.

2. Ciocârlie, H., Eleş, P., Balla, I., *Limbajele de programare PASCAL şi PASCAL CONCURENT*, Ed. Facla, Timişoara, 1985.

3. Creţu, V., *Structuri de date*, I.P. Timişoara, 1987.

4. Wirth, N., *Algorithms and data structures*, New Jersey, 1986.

5. * * * *PASCAL-LANGUAGE HISOFT*, Cheddington, G.B. 1984.

6. * * * *Hisoft Pascal 4TM Implementation Note*.

7. Dupont, R., Rollke, K., Szeliga, M., *Pascal auf dem ZX Spectrum*, Düsseldorf, 1984.

8. Czerwinski, *Programmieren in Pascal*, München 1982.

9. * * * *Pascal - manual de utilizare*, CTCET, 1982.

10. Fussinger, M., *Programierkurs Turbo Pascal*, Düsseldorf, 1988.

11. Scharf, I., *PASCAL für anfanger"*, R. Oldenburg Verlag Wien München, 1982.

In curs de apariție:

Colecția:

Biblioteca pentru elevi și studenți

- *Algoritmi și programe BASIC*
- *Culegere de probleme în Turbo-Pascal (la cerere cu floppy-disk)*
- *Limbajul C*
- *Culegere de programe în C (la cerere cu floppy-disk)*
- *BASIC între teorie și practică*
- *Sisteme de operare interactive*

Inițiere în informatică

ABC Info

- *Primii pași în programarea calculatoarelor*

Pentru specialiști

HIGH info

- *Limbajul PROLOG*
- *Metode și algoritmi de clasificare și recunoașterea formelor*

ISBN 973-95118-0-5